import mysql.connector

import tkinter as tk

from tkinter import ttk, messagebox

from PIL import Image, ImageTk

from tkinter import Listbox

from tkinter import \*

import re

import pygame

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import os

import math

import random

from datetime import datetime

from functools import partial

conn = mysql.connector.connect(

    host='localhost',

    user='root',

    password='root',

    database='project'  # Use the 'project' database

)

# Create a cursor object

cursor = conn.cursor()

# Create the 'items' table if it doesn't exist

cursor.execute('''

    CREATE TABLE IF NOT EXISTS items (

        itemno INT AUTO\_INCREMENT PRIMARY KEY,

        item\_name VARCHAR(255),

        quantity INT,

        unit\_price FLOAT,

        expiry\_date DATE,

        sale\_price FLOAT,

        gst FLOAT,

        net\_price FLOAT

    )

''')

root = tk.Tk()

root.title("Grocery Store Management")

root.geometry("1024x1024")

# Load a background image (you should replace 'xyz.png' with your actual image path)

bg\_image = tk.PhotoImage(file=r"C:\Users\babu\Desktop\project\intro.png")

bg\_label = tk.Label(root, image=bg\_image)

bg\_label.place(relwidth=1, relheight=1)

pygame.init()

# Load the music file

pygame.mixer.music.load(r"C:\Users\babu\Desktop\project\welcome.mp3")

# Play the music

pygame.mixer.music.play()

# Define a common style for widgets

common\_style = {"font": ("Arial", 25), "bg": "black","fg":"white", "padx": 10, "pady": 10}

frames = []

# Function to log in as an administrator

def login\_administrator():

    username = username\_entry.get()

    password = password\_entry.get()

    if username == "admin" and password == "admin":

        messagebox.showinfo("Success", "Administrator logged in successfully.")

        show\_admin\_menu()

    else:

        messagebox.showinfo("Error", "Invalid username or password.")

username\_entry=None

password\_entry=None

# Function to log in as a customer

customer\_username\_entry = None

customer\_password\_entry = None

customer\_email\_entry = None

security\_question\_entry = None

    # Add logic to authenticate customers here

# Function to sign up as a customer

def sign\_up\_customer():

    global customer\_username\_entry

    customer\_username = customer\_username\_entry.get()

    customer\_password = customer\_password\_entry.get()

    customer\_email = customer\_email\_entry.get()

    sec\_question = security\_question\_entry.get()

    # Check if the username meets the conditions

    if not re.search(r'[A-Z]', customer\_username):

        messagebox.showinfo("Error", "Username must contain at least one capital letter.")

        return

    # Check if the password meets the conditions

    if not (re.search(r'[A-Z]', customer\_password) and re.search(r'[!@#$%^&\*?<>/.,=+\_-]', customer\_password) and len(customer\_password) >= 8):

        messagebox.showinfo("Error", "Password must have at least one capital letter, one special character, and be at least 8 characters long.")

        return

    # Check if the email is not empty

    if not customer\_email:

        messagebox.showinfo("Error", "Please fill in the email field.")

        return

    if not sec\_question:

        messagebox.showinfo("Error", "Please fill in the security question field.")

        return

    # Check if the username is unique

    cursor.execute("SELECT \* FROM credentials WHERE USERNAME = %s", (customer\_username,))

    existing\_user = cursor.fetchone()

    if existing\_user:

        messagebox.showinfo("Error", "Username already exists. Please choose a different username.")

        return

    # If all conditions are met, insert customer data into the database

    cursor.execute("INSERT INTO credentials (USERNAME, PASSWORD, EMAIL,security\_question) VALUES (%s, %s, %s,%s)",

                   (customer\_username, customer\_password, customer\_email,sec\_question))

    conn.commit()

    messagebox.showinfo("Success", "Account created successfully. You can now log in as a customer.")

def login\_customer():

    global customer\_username

    global customer\_password

    customer\_username = customer\_username\_entry.get()

    customer\_password = customer\_password\_entry.get()

    # Check the credentials against the SQL database

    cursor.execute("SELECT \* FROM credentials WHERE USERNAME = %s AND PASSWORD = %s", (customer\_username, customer\_password))

    result = cursor.fetchone()

    if result:

        messagebox.showinfo("Success", "Customer logged in successfully.")

        show\_customer\_menu()

    else:

        messagebox.showinfo("Error", "Invalid username or password.")

email\_entry=None

new\_password\_entry=None

security1\_question\_entry=None

def forgot\_password():

    def reset():

        customer\_username = customer\_username\_entry.get()

        sec1\_question = security1\_question\_entry.get()

        customer\_password = new\_password\_entry.get()

        cursor.execute("SELECT username FROM credentials")

        result = cursor.fetchall()

        for i in result:

            if i[0] == customer\_username:

                cursor.execute("SELECT security\_question FROM credentials WHERE username = %s", (customer\_username,))

                result1 = cursor.fetchall()

                for j in result1:

                    if j[0] == sec1\_question:

                        if len(customer\_password) < 8 or not any(char.isupper() for char in customer\_password) or not any(char.isdigit() for char in customer\_password) or not any(char.isalnum() or char in '!@#$%^&\*()\_+' for char in customer\_password):

                            messagebox.showinfo("Error", "Invalid password format. Password must have at least 8 characters with at least one uppercase letter, one digit, and one special character.")

                        else:

                            cursor.execute("UPDATE credentials SET password = %s WHERE username = %s", (customer\_password, customer\_username))

                            messagebox.showinfo("Success", "Password reset success. You can now log in as a customer.")

                            root2.destroy()

                            return

                    else:

                        messagebox.showinfo("Error", "Incorrect security question answer.")

                        return

        messagebox.showinfo("Error", "Invalid username.")

    # GUI setup

    root2 = tk.Tk()

    root2.title("Password Reset")

    username\_label = tk.Label(root2, text="Username")

    username\_label.grid(row=0, column=0)

    customer\_username\_entry = tk.Entry(root2)

    customer\_username\_entry.grid(row=0, column=1)

    sec1\_label = tk.Label(root2, text="enter your school studied or enter your favourite fruit")

    sec1\_label.grid(row=1, column=0)

    security1\_question\_entry = tk.Entry(root2)

    security1\_question\_entry.grid(row=1, column=1)

    new\_password\_label = tk.Label(root2, text="New Password")

    new\_password\_label.grid(row=2, column=0)

    new\_password\_entry = tk.Entry(root2)

    new\_password\_entry.grid(row=2, column=1)

    reset\_button = tk.Button(root2, text="Reset", command=reset)

    reset\_button.grid(row=3, column=0, columnspan=2)

# You may want to start the Tkinter main loop

# You may want to start the Tkinter main loop

#

admin\_login\_frame=None

def admin\_login():

    global username\_entry

    global password\_entry

    admin\_login\_frame = tk.Frame(root, bg="white", bd=5)

    admin\_login\_frame.place(relx=0.5, rely=0.5, relwidth=0.2, relheight=0.2, anchor=tk.CENTER)

    username\_label = tk.Label(admin\_login\_frame, text="Username:", bg="white")

    username\_label.grid(row=0, column=0)

    username\_entry = tk.Entry(admin\_login\_frame)

    username\_entry.grid(row=0, column=1)

    password\_label = tk.Label(admin\_login\_frame, text="Password:", bg="white")

    password\_label.grid(row=1, column=0)

    password\_entry = tk.Entry(admin\_login\_frame, show="\*")

    password\_entry.grid(row=1, column=1)

    login\_button = tk.Button(admin\_login\_frame, text="Login as Admin", command=login\_administrator)

    login\_button.grid(row=2, columnspan=2)

# Create widgets for logging in as a customer

customer\_login\_frame=None

def customer\_signin():

    global customer\_login\_frame, customer\_username\_entry, customer\_password\_entry  # Updated variable names

    customer\_login\_frame = tk.Frame(root, bg="blue", bd=10)

    customer\_login\_frame.place(relx=0.5, rely=0.2, relwidth=0.4, relheight=0.2, anchor="n")

    # Create the entry widgets

    customer\_username\_entry = tk.Entry(customer\_login\_frame)

    customer\_username\_entry.grid(row=0, column=1)

    customer\_password\_entry = tk.Entry(customer\_login\_frame, show="\*")

    customer\_password\_entry.grid(row=1, column=1)

    customer\_login\_button = tk.Button(customer\_login\_frame, text="Login as Customer", command=login\_customer)

    customer\_login\_button.grid(row=2, columnspan=2)

    customer\_username\_label = tk.Label(customer\_login\_frame, text="Username:", bg="white")

    customer\_username\_label.grid(row=0, column=0)

    customer\_password\_label = tk.Label(customer\_login\_frame, text="Password:", bg="white")

    customer\_password\_label.grid(row=1, column=0)

    forgot\_pswd = tk.Button(customer\_login\_frame, text="forgot password", command=forgot\_password)

    forgot\_pswd.grid(row=3, columnspan=2)

# Define global variables for customer entry fields

new\_gst\_entry=None

def customer\_signup():

    global customer\_username\_entry, customer\_password\_entry, customer\_email\_entry, security\_question\_entry  # Updated variable names

    customer\_sign\_up\_frame = tk.Frame(root, bg="white", bd=5)

    customer\_sign\_up\_frame.place(relx=0.5, rely=0.8, relwidth=0.4, relheight=0.2, anchor="n")

    customer\_signup\_username\_label = tk.Label(customer\_sign\_up\_frame, text="Username:", bg="white")

    customer\_signup\_username\_label.grid(row=0, column=0)

    customer\_username\_entry = tk.Entry(customer\_sign\_up\_frame)

    customer\_username\_entry.grid(row=0, column=1)

    customer\_signup\_password\_label = tk.Label(customer\_sign\_up\_frame, text="Password:", bg="white")

    customer\_signup\_password\_label.grid(row=1, column=0)

    customer\_password\_entry = tk.Entry(customer\_sign\_up\_frame, show="\*")

    customer\_password\_entry.grid(row=1, column=1)

    customer\_email\_label = tk.Label(customer\_sign\_up\_frame, text="Email:", bg="white")

    customer\_email\_label.grid(row=2, column=0)

    customer\_email\_entry = tk.Entry(customer\_sign\_up\_frame)

    customer\_email\_entry.grid(row=2, column=1)

    security\_question\_label = tk.Label(customer\_sign\_up\_frame, text="enter your school studied or enter your favourite fruit", bg="white")

    security\_question\_label.grid(row=3, column=0)

    security\_question\_entry = tk.Entry(customer\_sign\_up\_frame)

    security\_question\_entry.grid(row=3, column=1)

    sign\_up\_button = tk.Button(customer\_sign\_up\_frame, text="Sign Up as Customer", command=sign\_up\_customer)

    sign\_up\_button.grid(row=4, columnspan=2)

    sign\_up\_button = tk.Button(customer\_sign\_up\_frame, text="Sign Up as Customer", command=sign\_up\_customer)

    sign\_up\_button.grid(row=4, columnspan=2)

login\_frame = ttk.Frame(root)

login\_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

b1 = tk.Button(login\_frame, text="Administrator Login", command=admin\_login, \*\*common\_style)

b2 = tk.Button(login\_frame, text="Customer Login", command=customer\_signin, \*\*common\_style)

b3 = tk.Button(login\_frame, text="Customer Signup", command=customer\_signup, \*\*common\_style)

b1.pack(pady=60)

b2.pack(pady=60)

b3.pack(pady=60)

gst\_entry=None

# Create widgets for the administrator's menu

def show\_admin\_menu():

   # Initialize pygame

    pygame.init()

   # Load the music file

    pygame.mixer.music.load(r"C:\Users\babu\Desktop\project\music.mp3")

   # Play the music

    pygame.mixer.music.play()

    global admin\_login\_frame

    if admin\_login\_frame is not None:

        admin\_login\_frame.destroy()

    login\_frame.destroy() # Destroy the login frame

    notebook = ttk.Notebook(root)

    notebook.pack(fill='both', expand='yes')

    # Create widgets for adding items

    def add\_item():

        item\_name = item\_name\_entry.get()

        quantity = quantity\_entry.get()

        unit\_price = unit\_price\_entry.get()

        item\_no = item\_no\_entry.get()

        exp\_date = exp\_date\_entry.get()

        sale\_price = sale\_price\_entry.get()

        gst=gst\_entry.get()

        query = "INSERT INTO items (itemno, item\_name, quantity, unit\_price, expiry\_date, sale\_price, gst) VALUES (%s, %s, %s, %s, %s, %s, %s)"

        values = (item\_no, item\_name, quantity, unit\_price, exp\_date, sale\_price,gst)

        try:

            print("Query:", query)

            print("Values:", values)

            cursor.execute(query, values)

            conn.commit()

            cursor.execute("update items SET net\_price=sale\_price+sale\_price\*gst/100")

            conn.commit()

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

        except mysql.connector.IntegrityError:

            messagebox.showinfo("Warning", "Item already exists. ")

    add\_item\_frame = ttk.Frame(root)

    notebook.add(add\_item\_frame, text='Add Item')

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\add entry.png")

# Add a label to display the image

    image\_label = ttk.Label(add\_item\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

# Create a custom style for labels, entry widgets, and the button

    custom\_style = ttk.Style()

    custom\_style.configure("Custom.TLabel", font=("Bernard MT condensed", 30))

    custom\_style.configure("Custom.TEntry", font=("Bernard MT condensed", 30))

    custom\_style.configure("Custom.TButton", font=("Bernard MT condensed", 30), background="black",foreground="green")

# Create labels with the custom style

    item\_name\_label = ttk.Label(add\_item\_frame, text="Product Name:", style="Custom.TLabel")

    item\_name\_label.grid(row=0, column=0, padx=10, pady=5)

    item\_name\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    item\_name\_entry.grid(row=0, column=1, padx=10, pady=5)

    quantity\_label = ttk.Label(add\_item\_frame, text="Quantity:", style="Custom.TLabel")

    quantity\_label.grid(row=1, column=0, padx=10, pady=5)

    quantity\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    quantity\_entry.grid(row=1, column=1, padx=10, pady=5)

    unit\_price\_label = ttk.Label(add\_item\_frame, text="Unit Price:", style="Custom.TLabel")

    unit\_price\_label.grid(row=2, column=0, padx=10, pady=5)

    unit\_price\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    unit\_price\_entry.grid(row=2, column=1, padx=10, pady=5)

    item\_no\_label = ttk.Label(add\_item\_frame, text="Item No:", style="Custom.TLabel")

    item\_no\_label.grid(row=3, column=0, padx=10, pady=5)

    item\_no\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    item\_no\_entry.grid(row=3, column=1, padx=10, pady=5)

    exp\_date\_label = ttk.Label(add\_item\_frame, text="Expiry Date (yy/mm/dd):", style="Custom.TLabel")

    exp\_date\_label.grid(row=4, column=0, padx=10, pady=5)

    exp\_date\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    exp\_date\_entry.grid(row=4, column=1, padx=10, pady=5)

    sale\_price\_label = ttk.Label(add\_item\_frame, text="Sale Price:", style="Custom.TLabel")

    sale\_price\_label.grid(row=5, column=0, padx=10, pady=5)

    sale\_price\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    sale\_price\_entry.grid(row=5, column=1, padx=10, pady=5)

    gst\_label = ttk.Label(add\_item\_frame, text="GST:", style="Custom.TLabel")

    gst\_label.grid(row=6, column=0, padx=10, pady=5)

    gst\_entry = ttk.Entry(add\_item\_frame, style="Custom.TEntry", width=30)  # Increase entry width

    gst\_entry.grid(row=6, column=1, padx=10, pady=5)

# Create and configure the "Add Item" button

    add\_button = ttk.Button(add\_item\_frame, text="Add Item", command=add\_item, style="Custom.TButton")

    add\_button.grid(row=7, columnspan=2, pady=10)

    # Create widgets for showing stock

    stock\_treeview = ttk.Treeview(notebook, columns=("itemno", "item\_name", "quantity", "unit\_price", "expiry\_date", "sale\_price","gst","net\_price"))

    notebook.add(stock\_treeview, text='Show Stock')

    button\_frame = ttk.Frame(stock\_treeview)

    button\_frame.grid(row=0, column=0, pady=10)

    # Define columns for the table

    stock\_treeview.heading("#1", text="Item No")

    stock\_treeview.heading("#2", text="Item Name")

    stock\_treeview.heading("#3", text="Quantity")

    stock\_treeview.heading("#4", text="Unit Price")

    stock\_treeview.heading("#5", text="Expiry Date")

    stock\_treeview.heading("#6", text="Sale Price")

    stock\_treeview.heading("#7", text="gst")

    stock\_treeview.heading("#8", text="net\_Price")

    # Set column widths

    stock\_treeview.column("#1", width=80)

    stock\_treeview.column("#2", width=100)

    stock\_treeview.column("#3", width=100)

    stock\_treeview.column("#4", width=100)

    stock\_treeview.column("#5", width=50)

    stock\_treeview.column("#6", width=50)

    stock\_treeview.column("#7", width=100)

    stock\_treeview.column("#8", width=100)

    # Function to populate the table with stock data

    def show\_stock():

        stock\_treeview.delete(\*stock\_treeview.get\_children())  # Clear the table

        cursor.execute("SELECT \* FROM items")

        result = cursor.fetchall()

        for row in result:

            stock\_treeview.insert("", "end", values=row)

    show\_stock\_button = ttk.Button(button\_frame, text="Show Stock", command=show\_stock)

    show\_stock\_button.grid(row=0, column=0, pady=10)

    style = ttk.Style()

    style.configure("Treeview.Heading", font=("Helvetica", 6))  # Increase the font size (12) as needed

# Increase the font size for cell values

      # Increase the font size (10) as needed

# Set the background color for the Treeview

    style.configure("Treeview", background="#D9E4FF")  # Change the color code to the color you want

    # Function to remove items from the database

    style\_options = {

        'font': ('Bernard MT Bold', 30),

        'background': 'white'}

    bill\_treeview = ttk.Treeview(notebook, columns=("bill\_id", "customer name","mode of payment", "item\_no", "item\_name", "unit\_price", "unit\_price", "sale\_price", "phone\_number", "address","purchase\_price","gst","net\_price"))

    notebook.add(bill\_treeview, text='Show billed items')

    button\_frame = ttk.Frame(bill\_treeview)

    button\_frame.grid(row=0, column=0, pady=10)

    bill\_treeview.heading("#1", text="Bill No")

    bill\_treeview.heading("#2", text="customer name")

    bill\_treeview.heading("#3", text="mode of payment")

    bill\_treeview.heading("#4", text="Item No")

    bill\_treeview.heading("#5", text="Item Name")

    bill\_treeview.heading("#6", text="unit\_price")

    bill\_treeview.heading("#7", text="quantity")

    bill\_treeview.heading("#8", text="Sale Price")

    bill\_treeview.heading("#9", text="Phone Number")

    bill\_treeview.heading("#10", text="Address")

    bill\_treeview.heading('#11',text='purchase\_price')

    bill\_treeview.heading('#12',text='gst')

    bill\_treeview.heading('#13',text='net\_price')

    # Set column widths

    bill\_treeview.column("#1", width=50)

    bill\_treeview.column("#2", width=150)

    bill\_treeview.column("#3", width=80)

    bill\_treeview.column("#4", width=150)

    bill\_treeview.column("#5", width=60)

    bill\_treeview.column("#6", width=80)

    bill\_treeview.column("#7", width=100)

    bill\_treeview.column("#8", width=80)

    bill\_treeview.column("#9", width=100)

    bill\_treeview.column("#10", width=150)

    bill\_treeview.column("#11", width=30)

    bill\_treeview.column("#12", width=100)

    bill\_treeview.column("#13", width=100)

    def show\_bills():

        bill\_treeview.delete(\*bill\_treeview.get\_children())  # Clear the table

        cursor.execute("SELECT \* FROM bills")

        result = cursor.fetchall()

        for row in result:

            bill\_treeview.insert("", "end", values=row)

    show\_stock\_button2 = ttk.Button(button\_frame, text="Show Bills", command=show\_bills)

    show\_stock\_button2.grid(row=1, column=0, pady=10)

    def remove\_item():

        item\_no = int(item\_no\_remove\_entry.get())

        qty = int(quantity\_remove\_entry.get())

        cursor.execute("SELECT quantity FROM items WHERE itemno = %s", (item\_no,))

        current\_qty = cursor.fetchone()

        if current\_qty and current\_qty[0] >= qty:

            cursor.execute("UPDATE items SET quantity = quantity - %s WHERE itemno = %s", (qty, item\_no))

            conn.commit()

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

        else:

            messagebox.showinfo("Warning", "Item does not exist or quantity is insufficient.")

    remove\_item\_frame = ttk.Frame(root)

    notebook.add(remove\_item\_frame, text='Remove Item')

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\remove.png")

# Add a label to display the image

    image\_label = ttk.Label(remove\_item\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

    item\_no\_remove\_label = ttk.Label(remove\_item\_frame, text="Item No to Remove", \*\*style\_options)

    item\_no\_remove\_label.grid(row=0, column=0, padx=10, pady=5)

    item\_no\_remove\_entry = ttk.Entry(remove\_item\_frame, \*\*style\_options)

    item\_no\_remove\_entry.grid(row=0, column=1, padx=10, pady=5)

    quantity\_remove\_label = ttk.Label(remove\_item\_frame, text="Quantity to Remove", \*\*style\_options)

    quantity\_remove\_label.grid(row=1, column=0, padx=10, pady=5)

    quantity\_remove\_entry = ttk.Entry(remove\_item\_frame, \*\*style\_options)

    quantity\_remove\_entry.grid(row=1, column=1, padx=10, pady=5)

    remove\_button = ttk.Button(remove\_item\_frame, text="Remove Item", command=remove\_item, style="Custom.TButton")

    remove\_button.grid(row=2, columnspan=2, pady=10)

    # Load the image

    def modify\_item():

        item\_no = item\_no\_modify\_entry.get()

        new\_quantity = new\_quantity\_entry.get()

        new\_unit\_price = new\_unit\_price\_entry.get()

        new\_exp\_date = new\_exp\_date\_entry.get()

        new\_sale\_price = new\_sale\_price\_entry.get()

        new\_gst=new\_gst\_entry.get()

        query = "UPDATE items SET quantity = %s, unit\_price = %s, expiry\_date = %s, sale\_price=%s,gst=%s WHERE itemno = %s"

        values = (new\_quantity, new\_unit\_price, new\_exp\_date, new\_sale\_price,new\_gst, item\_no)

        cursor.execute(query, values)

        conn.commit()

        messagebox.showinfo("Success", f"Item {item\_no} modified successfully.")

# Create widgets for modifying items

    modify\_item\_frame = ttk.Frame(root)

    notebook.add(modify\_item\_frame, text='Modify Item')

    xyz = ttk.Style()

    xyz.configure("custom.TLabel", font=("Bernard MT condensed", 30))

    xyz.configure("custom.TEntry", font=("Bernard MT condensed", 30))

    xyz.configure("custom.TButton", font=("Bernard MT condensed", 30), background="black",foreground="blue")

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\modify.png")

# Add a label to display the image

    image\_label = ttk.Label(modify\_item\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

    item\_no\_modify\_label = ttk.Label(modify\_item\_frame, text="Item No to Modify:",style="Custom.TLabel")

    item\_no\_modify\_label.grid(row=0, column=0, padx=10, pady=5)

    item\_no\_modify\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    item\_no\_modify\_entry.grid(row=0, column=1, padx=10, pady=5)

    new\_quantity\_label = ttk.Label(modify\_item\_frame, text="New Quantity:",style="Custom.TLabel")

    new\_quantity\_label.grid(row=1, column=0, padx=10, pady=5)

    new\_quantity\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    new\_quantity\_entry.grid(row=1, column=1, padx=10, pady=5)

    new\_unit\_price\_label = ttk.Label(modify\_item\_frame, text="New Unit Price:",style="Custom.TLabel")

    new\_unit\_price\_label.grid(row=2, column=0, padx=10, pady=5)

    new\_unit\_price\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    new\_unit\_price\_entry.grid(row=2, column=1, padx=10, pady=5)

    new\_exp\_date\_label = ttk.Label(modify\_item\_frame, text="New Expiry Date (yy/mm/dd):",style="Custom.TLabel")

    new\_exp\_date\_label.grid(row=3, column=0, padx=10, pady=5)

    new\_exp\_date\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    new\_exp\_date\_entry.grid(row=3, column=1, padx=10, pady=5)

    new\_sale\_price\_label = ttk.Label(modify\_item\_frame, text="New Sale Price:",style="Custom.TLabel")

    new\_sale\_price\_label.grid(row=4, column=0, padx=10, pady=5)

    new\_sale\_price\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    new\_sale\_price\_entry.grid(row=4, column=1, padx=10, pady=5)

    new\_gst\_label = ttk.Label(modify\_item\_frame, text="New gst:",style="Custom.TLabel")

    new\_gst\_label.grid(row=5, column=0, padx=10, pady=5)

    new\_gst\_entry = ttk.Entry(modify\_item\_frame,style="custom.TEntry")

    new\_gst\_entry.grid(row=5, column=1, padx=10, pady=5)

    modify\_button = ttk.Button(modify\_item\_frame, text="Modify Item", command=modify\_item,style="custom.TButton")

    modify\_button.grid(row=7, columnspan=2, pady=10)

        # Create widgets for graphing data

    def profit\_earned():

        cursor

        cursor.execute("select sum(total\_price)-sum(purchase\_price\*quantity) from bills")

        result = cursor.fetchall()

        k=result[0][0]

        cursor.execute("select sum(unit\_price\*quantity )from items where expiry\_date<=curdate() ")

        result1 = cursor.fetchall()

        if result1 and result1[0][0] is not None:

            l = result1[0][0]

        else:

            l = 0

        profit=k-l

        profit\_display=ttk.Label(profit\_frame,text=profit,style="custom.TLabel")

        profit\_display.grid(row=4, column=0, padx=100, pady=100)

    profit\_frame=ttk.Frame(root)

    notebook.add(profit\_frame, text='profit earned')

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\profit.png")

# Add a label to display the image

    image\_label = ttk.Label(profit\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

    ss=ttk.Button(profit\_frame,text="show profit",command=profit\_earned,style="custom.TButton")

    ss.grid(row=5,pady=10,columnspan=2)

    xyz = ttk.Style()

    xyz.configure("custom.TLabel", font=("Bernard MT condensed", 30))

    xyz.configure("custom.TEntry", font=("Bernard MT condensed", 30))

    xyz.configure("custom.TButton", font=("Bernard MT condensed", 30), background="black",)

    cursor.execute("SELECT SUM(quantity),item\_name FROM bills GROUP BY item\_name ORDER BY item\_name")

    result7 = cursor.fetchall()

    cursor.execute("SELECT DISTINCT item\_name FROM bills ORDER BY item\_name")

    result8 = cursor.fetchall()

    sales = [item[0] for item in result7]

    items = [item[0] for item in result8]

    def graphs1():

        fig = plt.figure(figsize = (7,5))

        axes = fig.add\_subplot(1,1,1)

        axes.set\_ylim(0, 300)

        palette = ['blue', 'red', 'green',

           'darkorange', 'maroon', 'black']

        plt.bar(items,sales,width=0.3,color=palette)

        plt.title('Profit Earned')

        plt.xlabel("ITEMS")

        plt.ylabel("SALES")

        plt.show()

    def graphs2():

        plt.plot(items,sales)

        plt.xlabel("ITEMS")

        plt.ylabel("SALES")

        plt.show()

    def graphs3():

        plt.pie(sales, labels=items)

        plt.title('Sales')

        plt.show()

    graph\_frame = ttk.Frame(root)

    notebook.add(graph\_frame, text='Graph Data')

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\graph.png")

# Add a label to display the image

    image\_label = ttk.Label(graph\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

    graph\_button1=ttk.Button(graph\_frame,text="Generate bargraph",command=graphs1,style="custom.TButton")

    graph\_button1.grid(row=0, columnspan=1, padx=150, pady=10)

    graph\_button2=ttk.Button(graph\_frame,text="Generate linechart",command=graphs2,style="custom.TButton")

    graph\_button2.grid(row=4,columnspan=2,padx=150,pady=70)

    graph\_button3=ttk.Button(graph\_frame,text="Generate piechart",command=graphs3,style="custom.TButton")

    graph\_button3.grid(row=8,columnspan=3,padx=150,pady=90)

    user\_treeview = ttk.Treeview(notebook, columns=("username", "password","email", "security question"))

    notebook.add(user\_treeview, text='Show users')

    user\_treeview.heading("#1", text="username")

    user\_treeview.heading("#2", text="password")

    user\_treeview.heading("#3", text="email")

    user\_treeview.heading("#4", text="security\_question")

    # Set column widths

    user\_treeview.column("#1", width=80)

    user\_treeview.column("#2", width=200)

    user\_treeview.column("#3", width=100)

    user\_treeview.column("#4", width=100)

    # Function to populate the table with stock data

    def show\_users():

        user\_treeview.delete(\*user\_treeview.get\_children())  # Clear the table

        cursor.execute("SELECT \* FROM credentials")

        result = cursor.fetchall()

        for row in result:

            user\_treeview.insert("", "end", values=row)

    user\_frame = ttk.Frame(user\_treeview)

    user\_frame.grid(row=0, column=0, pady=10)

    show\_user\_button = ttk.Button(user\_frame, text="Show users", command=show\_users)

    show\_user\_button.grid(row=1, column=0, pady=10)

    style = ttk.Style()

    style.configure("Treeview.Heading", font=("Helvetica", 18))

    # Increase the font size for cell values

    # Increase the font size (10) as needed

    # Set the background color for the Treeview

    style.configure("Treeview", background="#D9E4FF")  # Change the color code to the color you want

    delete\_username\_entry = None

    del\_frame = ttk.Frame(root)

    notebook.add(del\_frame, text='delete Users')

    xyzp= ttk.Style()

    xyzp.configure("custom.TLabel", font=("Bernard MT condensed", 30))

    xyzp.configure("custom.TEntry", font=("Bernard MT condensed", 30))

    xyzp.configure("custom.TButton", font=("Bernard MT condensed", 30), background="black",)

    def del1():

        delete\_username = delete\_username\_entry.get()

        cursor.execute("DELETE FROM credentials WHERE username=%s", (delete\_username,))

        conn.commit()

        messagebox.showinfo("deleted successfully")

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\users.png")

# Add a label to display the image

    image\_label = ttk.Label(del\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

    delete\_label = ttk.Label(del\_frame, text="enter userid to be deleted", style="custom.TLabel")

    delete\_username\_entry = ttk.Entry(del\_frame, style="custom.TEntry")

    delete\_button = ttk.Button(del\_frame, text="delete", command=del1, style="custom.TButton")

    delete\_label.grid(row=1, column=1,padx=150,pady=50)

    delete\_username\_entry.grid(row=4, column=1,pady=100,padx=100)

    delete\_button.grid(row=5, column=1,pady=150,padx=150)

    frames.append(admin\_login\_frame)

      # Add the admin menu frame to the frames list

# Create widgets for the customer portal

customer\_menu\_frame = None

login\_entry = None

customer\_phone\_number\_entry=None

customer\_address\_entry=None

def show\_customer\_menu():

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS bills (

        bill\_id INT AUTO\_INCREMENT PRIMARY KEY,

        customer\_name VARCHAR(255),

        mode\_of\_payment VARCHAR(255),

        item\_no INT,

        item\_name VARCHAR(255),

        unit\_price FLOAT,

        quantity INT,

        total\_price FLOAT,

        phone\_number varchar(10),

        address varchar(800),

        purchase\_price int,

        gst int,

        net\_price int

    )

''')

    current\_bill = []

    global customer\_login\_frame, frames, login\_entry

    notebook = ttk.Notebook(root)

    notebook.pack(fill='both', expand='yes')

    if customer\_login\_frame is not None:

        customer\_login\_frame.destroy()

    login\_frame.destroy()  # Destroy the login frame

    customer\_cart = {}

    def order\_groceries():

            current\_bill = []

# Initialize the Tkinter window

    billing\_frame = ttk.Frame(root)

    notebook.add(billing\_frame, text='Billing')

    image = PhotoImage(file=r"C:\users\babu\Desktop\project\bill.PNG")

# Add a label to display the image

    image\_label = ttk.Label(billing\_frame, image=image)

    image\_label.place(x=0, y=0, relwidth=1, relheight=1)

    image\_label.image = image

# Function to display the current bill

# Function to update the displayed bill

    def display\_customer\_info():

        bill\_text.config(state=tk.NORMAL)

        bill\_text.delete("1.0", tk.END)

        bill\_text.insert(tk.END, "\*" \* 100 + "\n")

        bill\_text.insert(tk.END, "Customer Name: {}\tMode of Payment: {}\n".format(customer\_name\_entry.get(), mode\_of\_payment\_entry.get()))

        bill\_text.insert(tk.END, "\*" \* 100 + "\n")

        bill\_text.config(state=tk.DISABLED)

# Function to display items in the bill

    def display\_items():

        bill\_text.config(state=tk.NORMAL)

        bill\_text.insert(tk.END, "Item No\t\tItem Name\t\t\tUnit Price\t\tQuantity\t\tTotal Price\t\tgst\tnet\_price\n")

        bill\_text.insert(tk.END, "\*" \* 105 + "\n")

        for item in current\_bill:

            item\_no, item\_name, unit\_price, quantity, total\_price,gst,net\_price= item

            bill\_text.insert(tk.END, f"{item\_no}\t\t{item\_name}\t\t\t{unit\_price}\t\t{quantity}\t\t{total\_price}\t\t{gst}\t{net\_price}\n")

            bill\_text.insert(tk.END, "\*" \* 105 + "\n")

        total\_amount = sum(item[6] for item in current\_bill)

        total\_amount\_label.config(text=f"Total Amount: {total\_amount:.2f}")

        bill\_text.config(state=tk.DISABLED)

# Function to update the displayed bill

    def update\_bill():

        display\_customer\_info()

        display\_items()

# Function to bill an item

  # Function to bill an item

    finalize\_stock\_update = False

    def bill\_item():

        item\_no = item\_no\_bill\_entry.get()

        quantity = quantity\_bill\_entry.get()

        customer\_name = customer\_name\_entry.get()

        mode\_of\_payment = mode\_of\_payment\_entry.get()

        if not item\_no or not quantity:

            messagebox.showinfo("Error", "Item No and Quantity are required.")

            return

        try:

            item\_no = int(item\_no)

            quantity = int(quantity)

            cursor.execute("SELECT item\_name, sale\_price, expiry\_date, gst FROM items WHERE itemno = %s", (item\_no,))

            result = cursor.fetchone()

            if result:

                item\_name, unit\_price, expiry\_date, gst = result

                cursor.execute("SELECT quantity FROM items WHERE itemno = %s", (item\_no,))

                current\_stock = cursor.fetchone()

                if current\_stock:

                    current\_stock = current\_stock[0]

                    total\_quantity\_in\_bill = sum(item[3] for item in current\_bill if item[0] == item\_no)

                    if current\_stock >= (quantity + total\_quantity\_in\_bill):

                        if expiry\_date > datetime.now().date():  # Check if the product is not expired

                            total\_price = unit\_price \* quantity

                            net\_price = total\_price \* gst / 100 + total\_price

                            current\_bill.append((item\_no, item\_name, unit\_price, quantity, total\_price, gst, net\_price))

                            update\_bill()

                        else:

                            messagebox.showerror('Expired', 'This product is Expired')

                    else:

                        messagebox.showinfo("Error", "Insufficient stock for this item.")

                else:

                    messagebox.showinfo("Error", "Item not found.")

            else:

                messagebox.showinfo("Error", "Item not found.")

        except ValueError:

            messagebox.showinfo("Error", "Invalid quantity. Please enter a valid number.")

# Function to finalize the bill

    def finalize\_bill():

        global finalize\_stock\_update

        customer\_name = customer\_name\_entry.get()

        mode\_of\_payment = mode\_of\_payment\_entry.get()

        customer\_phone\_number = customer\_phone\_number\_entry.get()

        customer\_address = customer\_address\_entry.get()

        if not customer\_phone\_number or not customer\_address:

            messagebox.showinfo("Error", "phone number address need to be filled .")

        else:

            pygame.init()

            # Load the music file

            pygame.mixer.music.load(r"C:\Users\babu\Desktop\project\confirm.mp3")

            # Play the music

            pygame.mixer.music.play()

            for item in current\_bill:

                item\_no, item\_name, unit\_price, quantity, total\_price, gst, net\_price = item

                cursor.execute("select unit\_price from items where itemno=%s", (item\_no,))

                price = cursor.fetchone()

                m = price[0]

                cursor.execute("select gst from items where itemno=%s", (item\_no,))

                tax = cursor.fetchone()

                t = tax[0]

                np = total\_price + (total\_price \* t / 100)

                pur\_price = unit\_price \* quantity

                cursor.execute(

                "INSERT INTO bills (customer\_name, mode\_of\_payment, item\_no, item\_name, unit\_price, quantity, total\_price,phone\_number,address,purchase\_price,gst,net\_price) VALUES (%s, %s, %s, %s, %s, %s, %s,%s,%s,%s,%s,%s)",

                (customer\_name, mode\_of\_payment, item\_no, item\_name, unit\_price, quantity, total\_price,

                customer\_phone\_number, customer\_address, m, t, np,))

                conn.commit()

            finalize\_stock\_update = True  # Set the flag to True to update stock

            current\_bill.clear()

            update\_bill()

            finalize\_stock\_update = False

    xyz = ttk.Style()

    xyz.configure("custom.TLabel", font=("Bernard MT condensed", 15),foreground="red")

    xyz.configure("custom.TEntry", font=("Bernard MT condensed", 30))

    xyz.configure("custom.TButton", font=("Bernard MT condensed", 15), background="black",foreground="green")

# Create UI elements

    customer\_name\_label = ttk.Label(billing\_frame, text="Customer Name:",style="custom.TLabel")

    customer\_name\_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")

    customer\_name\_entry = ttk.Entry(billing\_frame,style="custom.TEntry")

    customer\_name\_entry.grid(row=0, column=1, padx=10, pady=5, sticky="w")

    mode\_of\_payment\_label = ttk.Label(billing\_frame, text="Mode of Payment:",style="custom.TLabel")

    mode\_of\_payment\_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")

    mode\_of\_payment\_entry = ttk.Entry(billing\_frame,style="custom.TEntry")

    mode\_of\_payment\_entry.grid(row=1, column=1, padx=10, pady=5, sticky="w")

    item\_no\_bill\_label = ttk.Label(billing\_frame, text="Item No:",style="custom.TLabel")

    item\_no\_bill\_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")

    item\_no\_bill\_entry = ttk.Entry(billing\_frame)

    item\_no\_bill\_entry.grid(row=2, column=1, padx=10, pady=5, sticky="w")

    quantity\_bill\_label = ttk.Label(billing\_frame, text="Quantity:",style="custom.TLabel")

    quantity\_bill\_label.grid(row=3, column=0, padx=10, pady=5, sticky="w")

    quantity\_bill\_entry = ttk.Entry(billing\_frame)

    quantity\_bill\_entry.grid(row=3, column=1, padx=10, pady=5, sticky="w")

    bill\_button = ttk.Button(billing\_frame, text="Bill Item", command=bill\_item,style="custom.TButton")

    bill\_button.grid(row=4, columnspan=2, pady=10)

    bill\_text = tk.Text(billing\_frame, height=20, width=110)

    bill\_text.grid(row=5, columnspan=2, pady=10,sticky="nsew")

    bill\_text.config(state=tk.DISABLED)

    total\_amount\_label = ttk.Label(billing\_frame, text="Total Amount: 0.00",style="custom.TLabel")

    total\_amount\_label.grid(row=6, columnspan=2, pady=5)

    finalize\_button = ttk.Button(billing\_frame, text="Finalize Bill", command=finalize\_bill,style="custom.TButton")

    finalize\_button.grid(row=8, columnspan=2, pady=10)

    customer\_phone\_number\_label = ttk.Label(billing\_frame, text="Phone Number:",style="custom.TLabel")

    customer\_phone\_number\_label.grid(row=4, column=2, padx=40, pady=5)

    customer\_phone\_number\_entry = ttk.Entry(billing\_frame)

    customer\_phone\_number\_entry.grid(row=4, column=3, padx=45, pady=5)

    customer\_address\_label = ttk.Label(billing\_frame, text="Address:",style="custom.TLabel")

    customer\_address\_label.grid(row=5, column=2, padx=40, pady=5,)

    customer\_address\_entry = ttk.Entry(billing\_frame)

    customer\_address\_entry.grid(row=5, column=3, padx=45, pady=5)

    stock\_treeview = ttk.Treeview(notebook, columns=("itemno", "item\_name", "quantity",  "expiry\_date", "sale\_price","gst","net\_price"))

    notebook.add(stock\_treeview, text='Show Stock')

    button\_frame = ttk.Frame(stock\_treeview)

    button\_frame.grid(row=0, column=0, pady=10)

    # Define columns for the table

    stock\_treeview.heading("#1", text="Item No")

    stock\_treeview.heading("#2", text="Item Name")

    stock\_treeview.heading("#3", text="Quantity")

    stock\_treeview.heading("#4", text="Expiry Date")

    stock\_treeview.heading("#5", text="Sale Price")

    stock\_treeview.heading("#6", text="gst")

    stock\_treeview.heading("#7", text="net price")

    # Set column widths

    stock\_treeview.column("#1", width=100)

    stock\_treeview.column("#2", width=200)

    stock\_treeview.column("#3", width=100)

    stock\_treeview.column("#4", width=150)

    stock\_treeview.column("#5", width=100)

    stock\_treeview.column("#6", width=100)

    stock\_treeview.column("#7", width=100)

    # Function to populate the table with stock data

    def show\_stock():

        stock\_treeview.delete(\*stock\_treeview.get\_children())  # Clear the table

        cursor.execute("SELECT itemno,item\_name,quantity,expiry\_date,sale\_price,gst,net\_price FROM items")

        result = cursor.fetchall()

        for row in result:

            stock\_treeview.insert("", "end", values=row)

    show\_stock\_button = ttk.Button(button\_frame, text="Show Stock", command=show\_stock)

    show\_stock\_button.grid(row=0, column=0, pady=10)

    style = ttk.Style()

    style.configure("Treeview.Heading", font=("Helvetica", 18))  # Increase the font size (12) as needed

# Increase the font size for cell values

      # Increase the font size (10) as needed

# Set the background color for the Treeview

    style.configure("Treeview", background="#D9E4FF")  # Change the color code to the color you want

    pygame.init()

# Load the music file

    pygame.mixer.music.load(r"C:\Users\babu\Desktop\project\success.mp3")

# Play the music

    pygame.mixer.music.play()

root.mainloop()