```python
import tkinter as tk
from tkinter import ttk, messagebox
from tkcalendar import Calendar
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import sqlite3
from PIL import Image, ImageTk, ImageFilter

# Secure password hashing imports
import os
import hashlib
import binascii
import hmac

# For secure password hashing functions
def hash_password(password: str) -> str:
    """
    Hash a plaintext password using PBKDF2-HMAC-SHA256 with a random salt.
    Returns the hex-encoded salt+key.
    """
    salt = os.urandom(16)
    key = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 100000)
    return binascii.hexlify(salt + key).decode('utf-8')



def verify_password(stored_hash: str, candidate: str) -> bool:
    """
    Verify a candidate password against the stored hex-encoded salt+key.

    Supports legacy plain-text passwords: if the stored_hash isn't valid hex,
    falls back to a direct string comparison and can be rehashed later.
    """
    try:
        data = binascii.unhexlify(stored_hash.encode('utf-8'))
    except (binascii.Error, TypeError):
        # Legacy un-hashed password stored in DB
        return stored_hash == candidate

    salt, key = data[:16], data[16:]
    new_key = hashlib.pbkdf2_hmac('sha256', candidate.encode('utf-8'), salt, 100000)
    return hmac.compare_digest(new_key, key)

# Database Initialization
def initialize_database():
    conn = sqlite3.connect("marks2_data.db")
    cursor = conn.cursor()
    # Admin table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS admin (
            email TEXT PRIMARY KEY,
            password TEXT NOT NULL
        )
```

```
    ''')
    # Insert default admin with hashed password
    default_hash = hash_password('admin123')
    cursor.execute(
        "INSERT OR IGNORE INTO admin (email, password) VALUES (?, ?)",
        ('admin@example.com', default_hash)
    )
    # Marks table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS marks (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            module_code TEXT NOT NULL,
            module_name TEXT NOT NULL,
            cw1_marks INTEGER NOT NULL,
            cw2_marks INTEGER NOT NULL,
            cw3_marks INTEGER NOT NULL,
            student_id TEXT NOT NULL,
            student_name TEXT NOT NULL,
            gender TEXT NOT NULL,
            date_of_entry TEXT NOT NULL
        )
    ''')
    conn.commit()
    conn.close()

# Authenticate Admin
def authenticate():
    email = email_entry.get()
    password = password_entry.get()

    conn = sqlite3.connect("marks2_data.db")
    cursor = conn.cursor()
    cursor.execute("SELECT password FROM admin WHERE email = ?", (email,))
    row = cursor.fetchone()
    conn.close()

    if row and verify_password(row[0], password):
        messagebox.showinfo("Success", "Login successful!")
        show_main_app()
    else:
        messagebox.showerror("Error", "Invalid email or password.")

# Show Login View
def show_login():
    for widget in root.winfo_children():
        widget.destroy()

    # Path to your background image
    bg_image_path = "C:/Users/baich/Downloads/Premium Photo _ Double exposure
businessman working on digital tablet on white.jpeg"

    # Load and resize the image using Pillow
    bg_image = Image.open(bg_image_path)
    #bg_image = bg_image.filter(ImageFilter.GaussianBlur(radius=2))
    bg_image = bg_image.resize((root.winfo_screenwidth(), root.winfo_screenheight()),
Image.Resampling.LANCZOS)
```

```
    bg_photo = ImageTk.PhotoImage(bg_image)

    # Set the background image
    bg_label = tk.Label(root, image=bg_photo)
    bg_label.image = bg_photo  # Keep a reference to avoid garbage collection
    bg_label.place(relwidth=1, relheight=1)  # Stretch to cover the entire window

    # Create a frame for the login content
    login_frame = tk.Frame(root, bg="white", width=400, height=300,
highlightthickness=3, highlightbackground='skyblue')
    login_frame.place(relx=0.5, rely=0.5, anchor="center")  # Center the frame

    tk.Label(login_frame, text="Admin Login", font=("Arial", 20, "bold"),
bg="white").pack(pady=10)
    tk.Label(login_frame, text="Email", font=("Arial", 12, 'bold'),
bg="white").pack(anchor="w", padx=20, pady=5)

    # Entry for email
    global email_entry, password_entry
    email_entry = ttk.Entry(login_frame, width=30)
    email_entry.pack(pady=5, padx=20)

    tk.Label(
        login_frame,
        text="Password",
        font=("Arial", 12, 'bold'),
        bg="white"
    ).pack(anchor="w", padx=20, pady=5)

    # Entry for password
    password_entry = ttk.Entry(login_frame, width=30, show="*")
    password_entry.pack(pady=5, padx=20)

    # Login button
    login_button = tk.Button(
        login_frame,
        text="Login",
        command=authenticate,
        bg="lightskyblue",
        fg="black",
        font=("Arial", 12, "bold")
    )
    login_button.pack(pady=20)

    # Add hover effect for the login button
    login_button.default_bg = "lightskyblue"
    login_button.default_fg = "black"
    login_button.bind("<Enter>", on_enter)
    login_button.bind("<Leave>", on_leave)

    login_button.pack(pady=20)

# Add Logout Button
def logout():
```

```
    # Clear the current app and revert to login
    messagebox.showinfo("Logout", "You have been logged out.")
    show_login()

# Load data from the database
def load_data():
    conn = sqlite3.connect("marks2_data.db")
    df = pd.read_sql_query("SELECT * FROM marks", conn)
    conn.close()
    return df

# Save a new record to the database
def save_data(record):
    conn = sqlite3.connect("marks2_data.db")
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO marks (
            module_code, module_name, cw1_marks, cw2_marks, cw3_marks,
            student_id, student_name, gender, date_of_entry
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', (
        record["Module Code"],
        record["Module Name"],
        record["CW1 Marks"],
        record["CW2 Marks"],
        record["CW3 Marks"],
        record["Student ID"],
        record["Student Name"],
        record["Gender"],
        record["Date of Entry"]
    ))
    conn.commit()
    conn.close()

# Update an existing record in the database
def update_data(student_id, cw1, cw2, cw3):
    conn = sqlite3.connect("marks2_data.db")
    cursor = conn.cursor()
    cursor.execute('''
        UPDATE marks
        SET cw1_marks = ?, cw2_marks = ?, cw3_marks = ?
        WHERE student_id = ?
    ''', (cw1, cw2, cw3, student_id))
    conn.commit()
    conn.close()

# Validate marks input
def validate_marks(marks):
    try:
        marks = int(marks)
        return 0 <= marks <= 100
    except ValueError:
        return False

def on_enter(e):
    e.widget["background"] = "#346CB0"  # Hover background color
```

```
    e.widget["foreground"] = "black"     # Hover text color

def on_leave(e):
    e.widget["background"] = e.widget.default_bg  # Reset to default background
    e.widget["foreground"] = e.widget.default_fg  # Reset to default foreground
#lao bon




# Show Main App
def show_main_app():
    for widget in root.winfo_children():
        widget.destroy()

# Header Styling
    header_frame = tk.Frame(root, bg="cornflowerblue", height=80)
    header_frame.pack(fill="x")

    header_label = tk.Label(
    header_frame,
        text="Welcome to Marks Registration System",
        font=("Arial", 35, "bold"),
        fg="white",
        bg="cornflowerblue"
    )
    header_label.pack(pady=20)

# Create the Notebook for Tabs
    notebook_frame = tk.Frame(root, bg='white') # Set notebook frame background to
white
    notebook_frame.pack(fill="both")

    notebook = ttk.Notebook(root)
    notebook.pack(fill="both", expand=True)

# Tabs
    home_tab = ttk.Frame(notebook)
    input_tab = ttk.Frame(notebook)
    update_tab = ttk.Frame(notebook)
    view_tab = ttk.Frame(notebook)
    visualization_tab = ttk.Frame(notebook)

    notebook.add(home_tab, text="Home")
    notebook.add(input_tab, text="Input Marks")
    notebook.add(update_tab, text="Update Marks")
    notebook.add(view_tab, text="View Marks")
    notebook.add(visualization_tab, text="Visualization")

    # Add Logout Button
    logout_button = tk.Button(
        notebook,
        text="Logout",
        bg="blue",
        fg="black",
```

```
        font=("Arial", 12, "bold"),
        command=logout
    )
    logout_button.default_bg = "blue"
    logout_button.default_fg = "black"
    logout_button.bind("<Enter>", on_enter)
    logout_button.bind("<Leave>", on_leave)

    logout_button.place(relx=0.90, rely=0, anchor="ne")  # Adjust position as needed

# Additional Styling
    style = ttk.Style()
    style.theme_use("default")
    style.configure(
        "TNotebook",
        background="white",
        tabposition="n",
        borderwidth=0
    )
    style.configure(
        "TNotebook.Tab",
        font=("Arial", 12, "bold"),
        background="royalblue",
        foreground="white",
        padding=(10, 5),
    )
    style.map(
        "TNotebook.Tab",
        background=[("selected", "mediumblue")],
        foreground=[("selected", "white")],
    )

    # Set Tab Backgrounds to White
    style.configure("TFrame", background="white")  # Set all tab content areas to white

# Home Tab
    def update_home_tab():
        df = load_data()
        student_count = len(df["student_id"].unique())
        module_count = len(df["module_code"].unique())
        student_label.config(text=f"No. of Students: {student_count}")
        module_label.config(text=f"No. of Modules: {module_count}")

    student_label = tk.Label(
        home_tab,
        text="No. of Students: 0",
        font=("Arial", 16, "bold"),
        bg="white",
        fg="black"
    )
    student_label.pack(pady=10)

    module_label = tk.Label(
        home_tab,
        text="No. of Modules: 0",
        font=("Arial", 16, "bold"),
```

```
        bg="white",
        fg="black"
    )
    module_label.pack(pady=10)

    update_home_tab()

# Input Marks Tab
    def submit_input():
        module_code = module_code_entry.get()
        module_name = module_name_entry.get()
        cw1 = cw1_entry.get()
        cw2 = cw2_entry.get()
        cw3 = cw3_entry.get()
        student_id = student_id_entry.get()
        student_name = student_name_entry.get()
        gender = gender_var.get()
        date_of_entry = date_entry.get()

        if not (module_code and module_name and cw1 and cw2 and cw3 and student_id and
student_name and date_of_entry):
            messagebox.showerror("Error", "All fields are required.")
            return

        if not (validate_marks(cw1) and validate_marks(cw2) and validate_marks(cw3)):
            messagebox.showerror("Error", "Marks must be between 0 and 100.")
            return

        df = load_data()

        if not df.empty and ((df["student_id"] == student_id) & (df["module_code"] ==
module_code)).any():
            messagebox.showerror("Error", "Duplicate entry for this student and
module.")
            return

        new_data = {
            "Module Code": module_code,
            "Module Name": module_name,
            "CW1 Marks": int(cw1),
            "CW2 Marks": int(cw2),
            "CW3 Marks": int(cw3),
            "Student ID": student_id,
            "Student Name": student_name,
            "Gender": gender,
            "Date of Entry": date_of_entry,
        }

        save_data(new_data)
        messagebox.showinfo("Success", "Marks have been registered successfully!")
        reset_input()
        update_home_tab()

    def reset_input():
        module_code_entry.delete(0, tk.END)
```

```
        module_name_entry.delete(0, tk.END)
        cw1_entry.delete(0, tk.END)
        cw2_entry.delete(0, tk.END)
        cw3_entry.delete(0, tk.END)
        student_id_entry.delete(0, tk.END)
        student_name_entry.delete(0, tk.END)
        gender_var.set("Male")
        date_entry.delete(0, tk.END)

# Input Tab Content Container
    input_tab_content = tk.Frame(input_tab, bg="white")
    input_tab_content.grid(row=0, column=0, sticky="n")
    input_tab.columnconfigure(0, weight=1)

    input_tab_content.grid_columnconfigure(0, weight=1)
    input_tab_content.grid_columnconfigure(1, weight=1)

    def open_calendar():
        def select_date():
            date_entry.delete(0, tk.END)
            date_entry.insert(0, cal.get_date())
            cal_frame.destroy()

        cal_frame = tk.Frame(input_tab_content, bg="white")
        cal_frame.place(relx=0.5, rely=0, y=date_button.winfo_y() - 120, anchor="n")

        cal = Calendar(cal_frame, selectmode="day")
        cal.pack()

        ttk.Button(cal_frame, text="Choose Date", command=select_date).pack(pady=5)

    module_code_entry = ttk.Entry(input_tab_content, width=30)
    module_name_entry = ttk.Entry(input_tab_content, width=30)
    cw1_entry = ttk.Entry(input_tab_content, width=30)
    cw2_entry = ttk.Entry(input_tab_content, width=30)
    cw3_entry = ttk.Entry(input_tab_content, width=30)
    student_id_entry = ttk.Entry(input_tab_content, width=30)
    student_name_entry = ttk.Entry(input_tab_content, width=30)
    gender_var = tk.StringVar(value="Male")
    gender_menu = ttk.Combobox(input_tab_content, textvariable=gender_var,
values=["Male", "Female"], state="readonly")
    date_entry = ttk.Entry(input_tab_content, width=30)

    date_button = tk.Button(
        input_tab_content,
        text="Select Date",
        bg="cornflowerblue",
        fg="black",
        activebackground="lightsteelblue",
        activeforeground="black",
        font=("Arial", 10),
        command=open_calendar
    )

    date_button.default_bg = "cornflowerblue"
```

```
    date_button.default_fg = "black"
    date_button.bind("<Enter>", on_enter)
    date_button.bind("<Leave>", on_leave)

    submit_button = tk.Button(
        input_tab_content,
        text="Submit",
        bg="cornflowerblue",
        fg="black",
        activebackground="lightsteelblue",
        activeforeground="black",
        font=("Arial", 12),
        command=submit_input
    )

    submit_button.default_bg = "cornflowerblue"
    submit_button.default_fg = "black"
    submit_button.bind("<Enter>", on_enter)
    submit_button.bind("<Leave>", on_leave)

    reset_button = tk.Button(
        input_tab_content,
        text="Reset",
        bg="darkgrey",
        fg="black",
        activebackground="lightgrey",
        activeforeground="white",
        font=("Arial", 12),
        command=reset_input
    )

    reset_button.default_bg = "darkgrey"
    reset_button.default_fg = "black"
    reset_button.bind("<Enter>", on_enter)
    reset_button.bind("<Leave>", on_leave)

    widgets = [
        ("Module Code", module_code_entry),
        ("Module Name", module_name_entry),
        ("CW1 Marks", cw1_entry),
        ("CW2 Marks", cw2_entry),
        ("CW3 Marks", cw3_entry),
        ("Student ID", student_id_entry),
        ("Student Name", student_name_entry),
        ("Gender", gender_menu),
        ("Date of Entry", date_entry),
    ]

    for i, (label_text, widget) in enumerate(widgets):
        tk.Label(
            input_tab_content,
            text=label_text,
            bg="white",
            fg="black",
            font=("Arial", 12, "bold")
        ).grid(row=i, column=0, padx=10, pady=5, sticky="e")
```

```
        widget.grid(row=i, column=1, padx=10, pady=5, sticky="w")

    date_button.grid(row=8, column=2, padx=10, pady=5)
    submit_button.grid(row=len(widgets), column=0, padx=10, pady=10)
    reset_button.grid(row=len(widgets), column=1, padx=10, pady=10)

    # Update Marks Tab
    def search_update():
        student_id = update_student_id_entry.get()

        if not student_id:
            messagebox.showerror("Error", "Student ID is required.")
            return

        conn = sqlite3.connect("marks2_data.db")
        conn.row_factory = sqlite3.Row  # Enable named column access
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM marks WHERE student_id = ?", (student_id,))
        record = cursor.fetchone()
        conn.close()

        if not record:
            messagebox.showerror("Error", "Record not found.")
            return

        # Show all fields and populate them with data
        update_student_name_label.grid()
        update_student_name_entry.grid()
        update_module_code_label.grid()
        update_module_code_entry.grid()
        update_cw1_label.grid()
        update_cw1_entry.grid()
        update_cw2_label.grid()
        update_cw2_entry.grid()
        update_cw3_label.grid()
        update_cw3_entry.grid()
        update_button.grid()

        update_student_name_entry.delete(0, tk.END)
        update_module_code_entry.delete(0, tk.END)
        update_cw1_entry.delete(0, tk.END)
        update_cw2_entry.delete(0, tk.END)
        update_cw3_entry.delete(0, tk.END)

        # Populate fields using column names
        update_student_name_entry.insert(0, record["student_name"])
        update_module_code_entry.insert(0, record["module_code"])
        update_cw1_entry.insert(0, record["cw1_marks"])
        update_cw2_entry.insert(0, record["cw2_marks"])
        update_cw3_entry.insert(0, record["cw3_marks"])

        update_student_name_entry.configure(state="readonly")
        update_module_code_entry.configure(state="readonly")
```

```
    def update_record():
        student_id = update_student_id_entry.get()
        cw1 = update_cw1_entry.get()
        cw2 = update_cw2_entry.get()
        cw3 = update_cw3_entry.get()

        # Validate marks input
        if not (validate_marks(cw1) and validate_marks(cw2) and validate_marks(cw3)):
            messagebox.showerror("Error", "Marks must be between 0 and 100.")
            return

        try:
            conn = sqlite3.connect("marks2_data.db")
            cursor = conn.cursor()

            # Check if the record exists
            cursor.execute("SELECT * FROM marks WHERE student_id = ?", (student_id,))
            record = cursor.fetchone()

            if not record:
                messagebox.showerror("Error", "Record not found.")
                return

            # Update the record in the database
            cursor.execute("""
                UPDATE marks
                SET cw1_marks = ?, cw2_marks = ?, cw3_marks = ?
                WHERE student_id = ?
            """, (int(cw1), int(cw2), int(cw3), student_id))
            conn.commit()

            messagebox.showinfo("Success", "Marks have been updated successfully!")
            update_home_tab()  # Refresh the home tab stats
        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}")
        finally:
            conn.close()

    # Create a container frame for the Update Marks content
    update_tab_content = tk.Frame(update_tab, bg="white")
    update_tab_content.grid(row=0, column=0, sticky="n")
    update_tab.columnconfigure(0, weight=1)
    update_tab.rowconfigure(0, weight=1)

    update_tab_content.grid_columnconfigure(0, weight=1)
    update_tab_content.grid_columnconfigure(1, weight=1)

    # Widgets for Update Tab
    update_student_id_label = tk.Label(
        update_tab_content,
        text="Student ID",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
```

```
    update_student_id_entry = ttk.Entry(update_tab_content, width=30)
    search_button = tk.Button(
        update_tab_content,
        text="Search",
        bg="cornflowerblue",
        fg="black",
        activebackground="lightsteelblue",
        activeforeground="black",
        font=("Arial", 12),
        command=search_update
    )
    search_button.default_bg = "cornflowerblue"
    search_button.default_fg = "black"
    search_button.bind("<Enter>", on_enter)
    search_button.bind("<Leave>", on_leave)

    update_student_name_label = tk.Label(
        update_tab_content,
        text="Student Name",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
    update_student_name_entry = ttk.Entry(update_tab_content, width=30)

    update_module_code_label = tk.Label(
        update_tab_content,
        text="Module Code",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
    update_module_code_entry = ttk.Entry(update_tab_content, width=30)

    update_cw1_label = tk.Label(
        update_tab_content,
        text="CW1 Marks",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
    update_cw1_entry = ttk.Entry(update_tab_content, width=30)

    update_cw2_label = tk.Label(
        update_tab_content,
        text="CW2 Marks",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
    update_cw2_entry = ttk.Entry(update_tab_content, width=30)

    update_cw3_label = tk.Label(
        update_tab_content,
        text="CW3 Marks",
        bg="white",
```

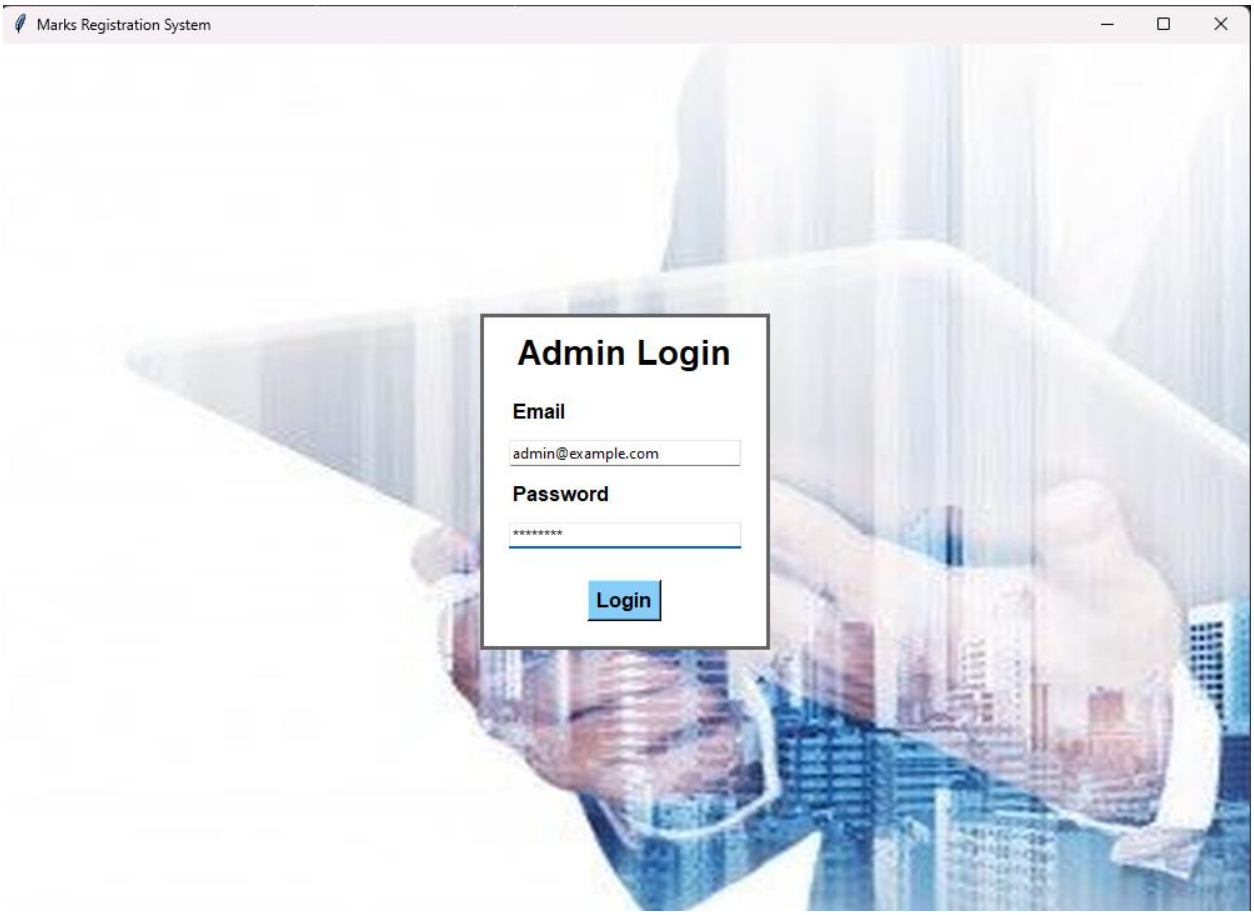
```
        fg="black",
        font=("Arial", 12, "bold")
    )
    update_cw3_entry = ttk.Entry(update_tab_content, width=30)

    update_button = tk.Button(
        update_tab_content,
        text="Update",
        bg="cornflowerblue",
        fg="black",
        activebackground="lightsteelblue",
        activeforeground="black",
        font=("Arial", 12),
        command=update_record
    )
    update_button.default_bg = "cornflowerblue"
    update_button.default_fg = "black"
    update_button.bind("<Enter>", on_enter)
    update_button.bind("<Leave>", on_leave)

    # Layout
    update_student_id_label.grid(row=0, column=0, padx=10, pady=5, sticky="e")
    update_student_id_entry.grid(row=0, column=1, padx=10, pady=5, sticky="w")
    search_button.grid(row=1, column=1, padx=10, pady=10)

    update_student_name_label.grid(row=2, column=0, padx=10, pady=5, sticky="e")
    update_student_name_entry.grid(row=2, column=1, padx=10, pady=5, sticky="w")
    update_module_code_label.grid(row=3, column=0, padx=10, pady=5, sticky="e")
    update_module_code_entry.grid(row=3, column=1, padx=10, pady=5, sticky="w")
    update_cw1_label.grid(row=4, column=0, padx=10, pady=5, sticky="e")
    update_cw1_entry.grid(row=4, column=1, padx=10, pady=5, sticky="w")
    update_cw2_label.grid(row=5, column=0, padx=10, pady=5, sticky="e")
    update_cw2_entry.grid(row=5, column=1, padx=10, pady=5, sticky="w")
    update_cw3_label.grid(row=6, column=0, padx=10, pady=5, sticky="e")
    update_cw3_entry.grid(row=6, column=1, padx=10, pady=5, sticky="w")
    update_button.grid(row=7, column=1, padx=10, pady=10)

    # Initially hide all fields except for Student ID and Search button
    update_student_name_label.grid_remove()
    update_student_name_entry.grid_remove()
    update_module_code_label.grid_remove()
    update_module_code_entry.grid_remove()
    update_cw1_label.grid_remove()
    update_cw1_entry.grid_remove()
    update_cw2_label.grid_remove()
    update_cw2_entry.grid_remove()
    update_cw3_label.grid_remove()
    update_cw3_entry.grid_remove()
    update_button.grid_remove()

    # View Marks Tab
    def view_records():
        student_id = view_student_id_entry.get()

        if not student_id:
            messagebox.showerror("Error", "Student ID is required.")
```

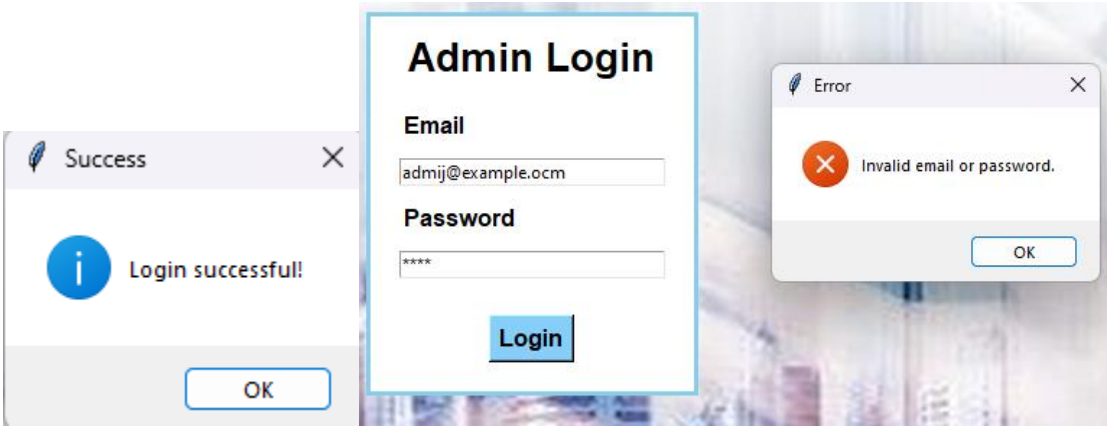
```
            return

        conn = sqlite3.connect("marks2_data.db")
        cursor = conn.cursor()
        cursor.execute("SELECT module_code, student_name, cw1_marks, cw2_marks,
cw3_marks FROM marks WHERE student_id = ?", (student_id,))
        records = cursor.fetchall()
        conn.close()

        if not records:
            messagebox.showinfo("Info", "No records found for the given Student ID.")
            return

        for row in tree.get_children():
            tree.delete(row)

        for record in records:
            total = record[2] + record[3] + record[4]
            tree.insert("", tk.END, values=(
                record[0],  # Module Code
                record[1],  # Student Name
                record[2],  # CW1 Marks
                record[3],  # CW2 Marks
                record[4],  # CW3 Marks
                total       # Total Marks
            ))

    # Create a container frame for the View Marks content
    view_tab_content = tk.Frame(view_tab, bg="white")
    view_tab_content.grid(row=0, column=0, sticky="n")  # Place it at the top-center of
the tab
    view_tab.columnconfigure(0, weight=1)  # Center horizontally
    view_tab.rowconfigure(0, weight=1)  # Center vertically

    view_tab_content.grid_columnconfigure(0, weight=1)  # Center labels
    view_tab_content.grid_columnconfigure(1, weight=1)  # Center widgets

    # Widgets for View Tab
    view_student_id_label = tk.Label(
        view_tab_content,
        text="Student ID",
        bg="white",
        fg="black",
        font=("Arial", 12, "bold")
    )
    view_student_id_entry = ttk.Entry(view_tab_content, width=30)
    view_button = tk.Button(
        view_tab_content,
        text="View",
        bg="cornflowerblue",
        fg="black",
        activebackground="lightsteelblue",
        activeforeground="black",
        font=("Arial", 12),
        command=view_records
```

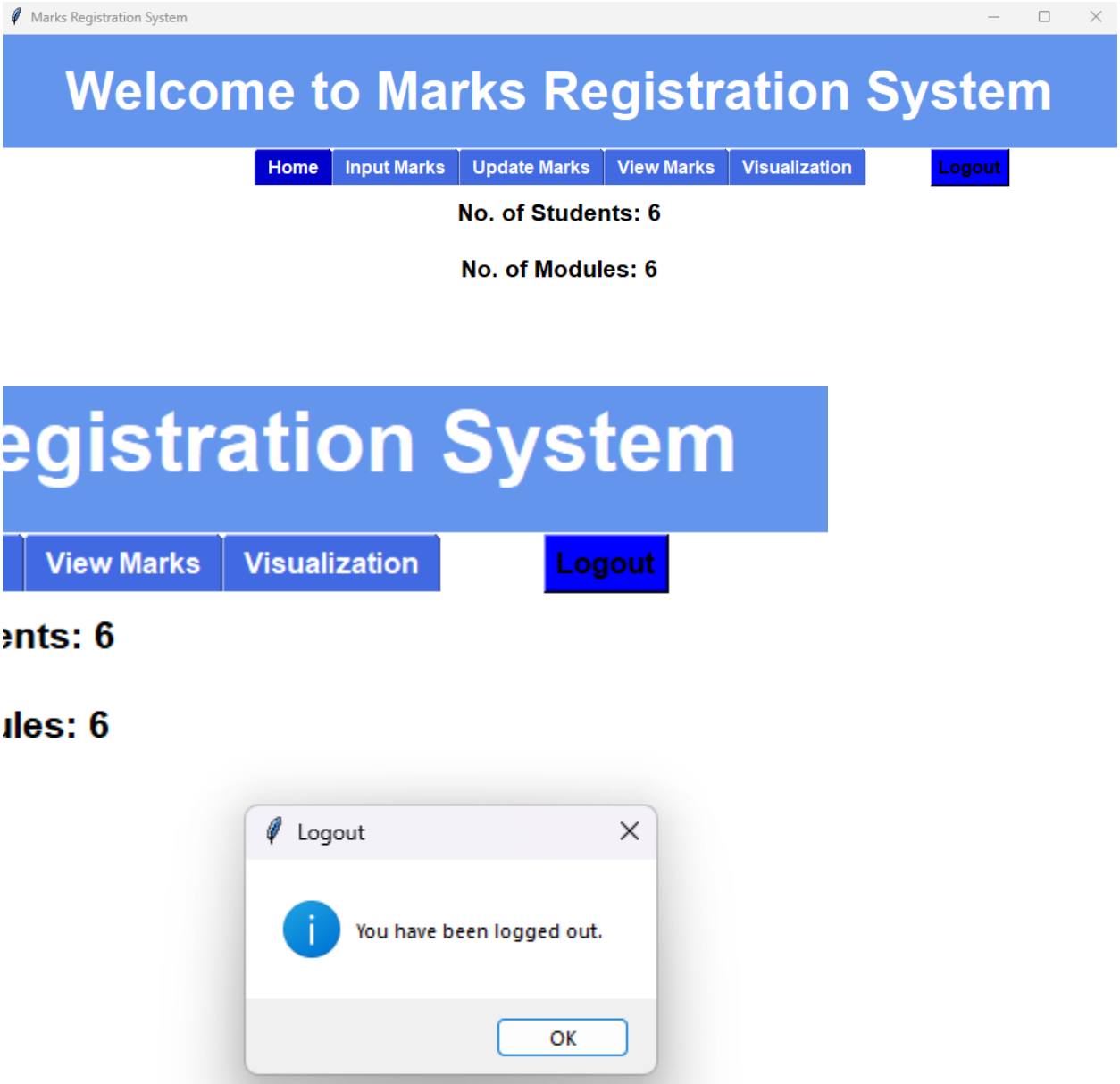
```
    )
    view_button.default_bg = "cornflowerblue"
    view_button.default_fg = "black"
    view_button.bind("<Enter>", on_enter)
    view_button.bind("<Leave>", on_leave)

    # Layout for input and button
    view_student_id_label.grid(row=0, column=0, padx=10, pady=5, sticky="e")
    view_student_id_entry.grid(row=0, column=1, padx=10, pady=5, sticky="w")
    view_button.grid(row=1, column=1, padx=10, pady=10, sticky="w")

    # Treeview for displaying records
    style = ttk.Style()
    style.configure(
        "Treeview.Heading",
        font=("Arial", 12, "bold"),
        background="blue",
        foreground="white",
        padding=(5, 5)
    )

    style.configure(
        "Treeview",
        background="white",
        foreground="black",
        rowheight=25,
        fieldbackground="white"
    )
    style.map(
        "Treeview",
        background=[("selected", "lightsteelblue")],
        foreground=[("selected", "black")]
    )

    columns = ("Module Code", "Student Name", "CW1 Marks", "CW2 Marks", "CW3 Marks",
"Total Marks")
    tree = ttk.Treeview(view_tab_content, columns=columns, show="headings",
style="Treeview")

    for col in columns:
        tree.heading(col, text=col, anchor='center')
        tree.column(col, width=150, anchor='center')

    tree.grid(row=2, column=0, columnspan=2, pady=10)

    # Visualization Tab
    def visualization():
        conn = sqlite3.connect("marks2_data.db")
        df = pd.read_sql_query("SELECT module_code, cw1_marks, cw2_marks, cw3_marks
FROM marks", conn)
        conn.close()

        if df.empty:
            messagebox.showinfo("Info", "No data available for visualization.")
            return
```

```
        # Clear existing plots
        for widget in visualization_tab.winfo_children():
            widget.destroy()

        # Define custom colors
        custom_colors = ['blue', 'mediumblue', 'royalblue']

        # Visualization Frame for Layout
        visualization_frame = tk.Frame(visualization_tab, bg="white")
        visualization_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
        visualization_tab.grid_columnconfigure(0, weight=1)  # Center the frame
horizontally
        visualization_tab.grid_rowconfigure(0, weight=1)  # Center the frame vertically

        # Add two columns in the frame for the graphs
        visualization_frame.grid_columnconfigure(0, weight=1)  # Center the first graph
        visualization_frame.grid_columnconfigure(1, weight=1)  # Center the second
graph

        # Pie Chart: Marks Distribution
        total_cw1 = df["cw1_marks"].sum()
        total_cw2 = df["cw2_marks"].sum()
        total_cw3 = df["cw3_marks"].sum()

        pie_fig, pie_ax = plt.subplots(figsize=(5, 5))
        pie_ax.pie(
            [total_cw1, total_cw2, total_cw3],
            labels=["CW1", "CW2", "CW3"],
            autopct="%1.1f%%",
            startangle=90,
            colors=custom_colors
        )
        pie_ax.set_title("Marks Distribution")

        pie_canvas = FigureCanvasTkAgg(pie_fig, master=visualization_frame)
        pie_canvas.draw()
        pie_canvas.get_tk_widget().grid(row=0, column=0, padx=10, pady=10, sticky="n")

        # Bar Chart: Average Marks by Module
        avg_marks = df.groupby("module_code")[["cw1_marks", "cw2_marks",
"cw3_marks"]].mean()

        bar_fig, bar_ax = plt.subplots(figsize=(7, 5))
        avg_marks.plot(kind="bar", ax=bar_ax, color=custom_colors)
        bar_ax.set_title("Average Marks by Module")
        bar_ax.set_ylabel("Average Marks")
        bar_ax.set_xlabel("Module Code")

        bar_canvas = FigureCanvasTkAgg(bar_fig, master=visualization_frame)
        bar_canvas.draw()
        bar_canvas.get_tk_widget().grid(row=0, column=1, padx=10, pady=10, sticky="n")

    visualization()
```

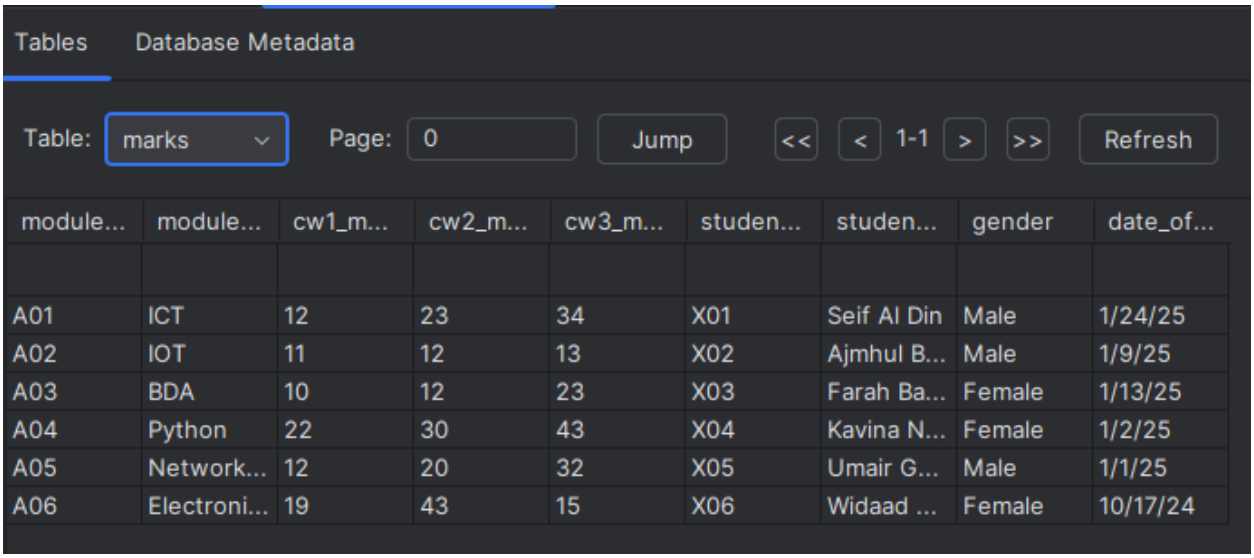
```python
# Create the main application window
if __name__ == "__main__":
    initialize_database()
    root = tk.Tk()
    root.title("Marks Registration System")
    root.geometry("1000x700")
    root.configure(bg="white")

    show_login()

    try:
        root.mainloop()  # Start the Tkinter event loop
    except KeyboardInterrupt:
        pass  # Suppress the KeyboardInterrupt error
```


Marks Registration System
Admin Login
Email
admin@example.com
Password
****
Login


Success
Login successful!
OK
Admin Login
Email
admij@example.ocm
Password
****
Login
Error
Invalid email or password.
OK


Marks Registration System
Welcome to Marks Registration System
Home
Input Marks
Update Marks
View Marks
Visualization
Logout
No. of Students: 6
No. of Modules: 6
egistration System
View Marks
Visualization
Logout
ents: 6
ules: 6
Logout
You have been logged out.
OK

marks2_data.db



module...	module...	cw1_m...	cw2_m...	cw3_m...	studen...	studen...	gender	date_of...
A01	ICT	12	23	34	X01	Seif Al Din	Male	1/24/25
A02	IOT	11	12	13	X02	Ajmhul B...	Male	1/9/25
A03	BDA	10	12	23	X03	Farah Ba...	Female	1/13/25
A04	Python	22	30	43	X04	Kavina N...	Female	1/2/25
A05	Network...	12	20	32	X05	Umair G...	Male	1/1/25
A06	Electroni...	19	43	15	X06	Widaad ...	Female	10/17/24