Project Assessment Documentation On

"Water Reminder App"

For the Assessment of

MCA First Semester (Division- I)

In the Course

Problem Solving with Python Lab (MCP544-3)

Developed by

Rohit Kawale (40)

Uday Rana (54)

Under the Guidance of

Prof. Pravin Y. Karmore

Assistant Professor, RCOEM

Department of Computer Application
Shri Ramdeobaba College of Engineering Management
Nagpur

INDEX

INTRODUCTION

The provided Python script is a graphical user interface (GUI) application built using the Tkinter library. This application is designed for user profile management, including user registration, profile updates, and viewing user profiles. Additionally, the script incorporates features for scheduling hydration reminders via email. tasks.

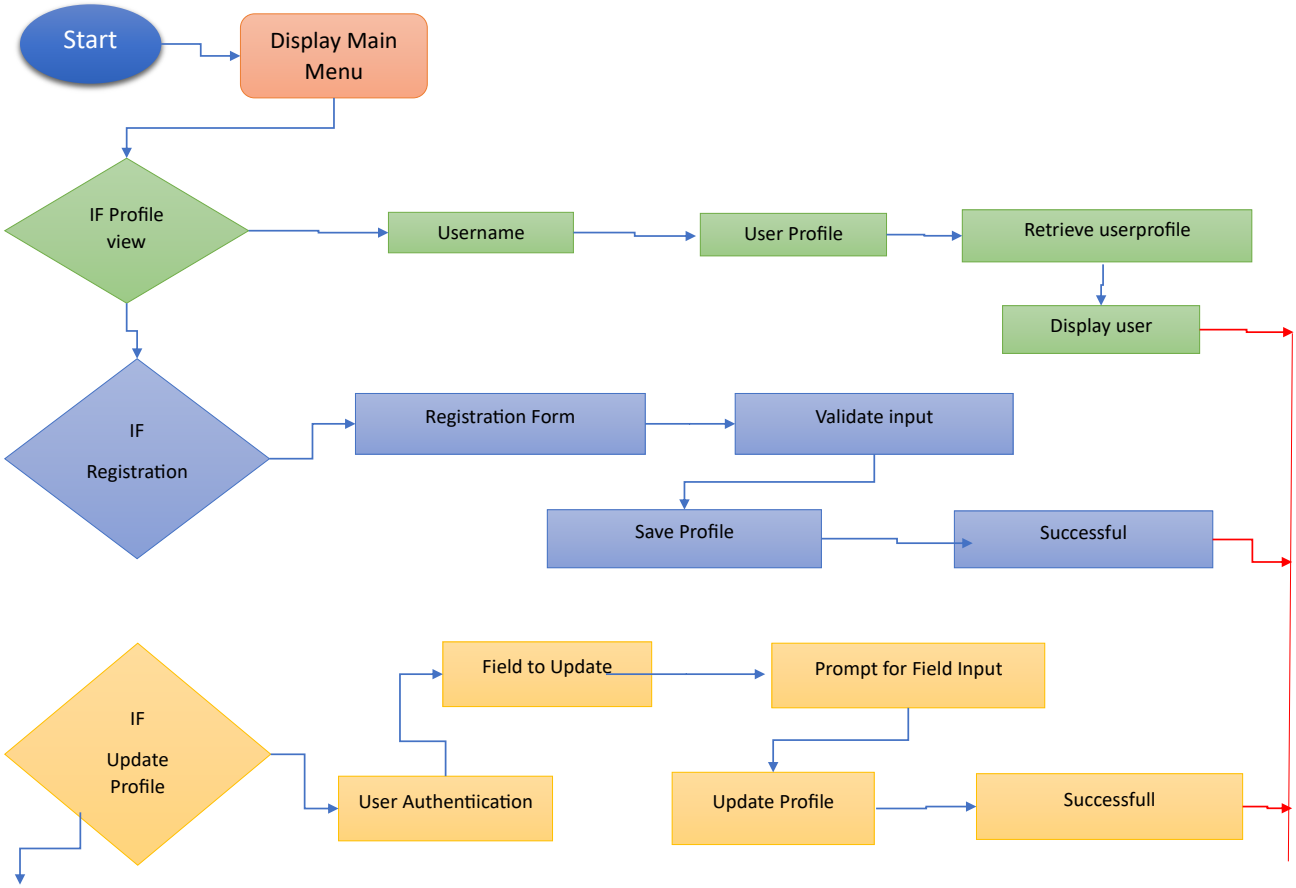
AIM AND OBJECTIVE

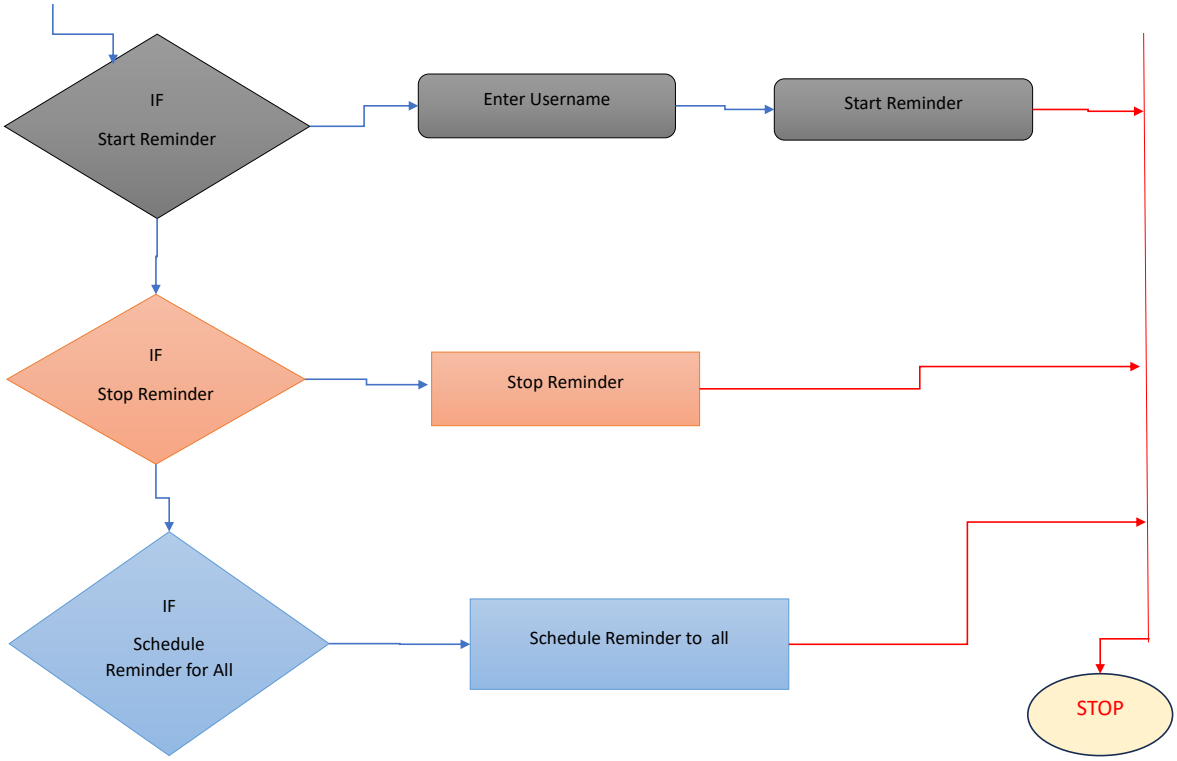
AIM: The aim of the script is to offer users a convenient way to manage their profiles, register new accounts, update profile information, and receive periodic hydration reminders.

Objectives:

1. **User Registration:** Allow users to register by providing a username, email, password, goal, age, height, and weight.
2. **User Authentication:** Authenticate users during profile updates to ensure security.
3. **Profile Updates:** Provide functionality for users to update various profile attributes such as username, email, goal, etc.
4. **Profile Viewing:** Allow users to view their profiles, displaying information such as user ID, username, email, goal, weight, and height.
5. **Hydration Reminders:** Implement a reminder system to send periodic hydration reminders to users via email.
6. **Scheduling Reminders:** Enable users to schedule reminders for specific times during the day.
7. **User Interface:** Create an intuitive and visually appealing GUI for a seamless user experience.

FLOW CHART


Start
Display Main Menu
IF Profile view
Username
User Profile
Retrieve userprofile
Display user
IF Registration
Registration Form
Validate input
Save Profile
Successful
IF Update Profile
Field to Update
Prompt for Field Input
User Authentiication
Update Profile
Successfull


IF
Start Reminder
Enter Username
Start Reminder
IF
Stop Reminder
Stop Reminder
IF
Schedule
Reminder for All
Schedule Reminder to all
STOP

CODE

```
import tkinter as tk
from tkinter import messagebox
import mysql.connector
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
import schedule
import json
import os
import time
import data.database as db
import threading
from tkinter import ttk  # Import themed widgets


# Global variables
username_entry = None
password_entry = None
profile_frame = None
registration_frame = None
update_frame = None
reminder_job = None  # Global variable to hold the scheduled job


# Functions for GUI actions
def switch_to_registration():
    profile_frame.pack_forget()
    update_frame.pack_forget()
    registration_frame.pack()

def switch_to_update():
    profile_frame.pack_forget()
    registration_frame.pack_forget()
    update_frame.pack()

def switch_to_profile():
    registration_frame.pack_forget()
    update_frame.pack_forget()
    profile_frame.pack()

def register_user(username_entry, email_entry, password_entry, goal_entry, age_entry,
height_entry, weight_entry):
    username = username_entry.get()
    email = email_entry.get()
    password = password_entry.get()  # Corrected the order here
    goal = goal_entry.get()
```

```
    age = age_entry.get()
    height = height_entry.get()
    weight = weight_entry.get()

    if username and password and email:  # Adjusted the order to ensure correct variables are
checked
        db.save_user_profile(username, email, password, goal, age, height, weight)
        messagebox.showinfo("Registration", "User registered successfully!")
    else:
        messagebox.showinfo("Registration", "Please enter username, email, and password!")



def authenticate_user(username_entry_auth,password_entry_auth):

    username = username_entry_auth.get()
    password = password_entry_auth.get()
    print(username)
    print(password)
    user_profile = db.get_user_profile(username)
    print(user_profile)

    # if user_profile and user_profile[3] == password:
    if user_profile:
        switch_to_update()
        create_update_widgets()
    else:
        messagebox.showinfo("Authentication", "Invalid username or password!")


def create_update_widgets():
    # Dropdown menu for selecting profile attributes
    attribute_list = ["Username", "Email", "Goal", "Age", "Height", "Weight"]
    selected_attribute = tk.StringVar()
    attribute_dropdown = tk.OptionMenu(update_frame, selected_attribute, *attribute_list)
    attribute_dropdown.pack()

    # Input field for new value
    new_value_entry = tk.Entry(update_frame)
    new_value_entry.pack()

    # Button to update profile attribute
    update_button = tk.Button(update_frame, text="Update", command=lambda:
update_attribute(selected_attribute.get(), new_value_entry.get()))
    update_button.pack()


def update_attribute(selected_attribute, new_value):
    username = username_entry.get()
```

```
    password = password_entry.get()
    user_profile = db.get_user_profile(username)
    print(user_profile,"helo")

    if user_profile and user_profile['password'] == password:  # Assuming password is stored
in the 'password' field
        if selected_attribute == "Username":
            db.update_user_profile(username, new_username=new_value)
            messagebox.showinfo("Update", "Username updated!")
        elif selected_attribute == "Email":
            db.update_user_profile(username, new_email=new_value)
            messagebox.showinfo("Update", "Email updated!")
        elif selected_attribute == "Goal":
            db.update_user_profile(username, new_goal=new_value)
            messagebox.showinfo("Update", "Goal updated!")
        # ... Repeat for other attributes

    else:
        messagebox.showinfo("Update", "Authentication failed. Please log in again.")


def get_profile():
    username = username_entry.get()
    if username:
        data = db.get_user_profile(username)
        if data:
            messagebox.showinfo("Profile Details", f"User ID: {data['user_id']}\n"
                                f"Username: {data['username']}\n"
                                f"Email: {data['email']}\n"
                                f"Goal: {data['goal']}\n"
                                f"Weight: {data['weight']}\n"
                                f"Height: {data['height']}")
        else:
            messagebox.showinfo("Profile Details", "No user with this username!")
    else:
        messagebox.showinfo("Profile Details", "Please enter a username!")

def read_config_from_json(file_name):
    script_directory = os.path.dirname(os.path.realpath(__file__))
    file_path = os.path.join(script_directory, 'data', file_name)

    with open(file_path, 'r') as file:
        config_data = json.load(file)

    return config_data


config_data = read_config_from_json('hydration_goals.json')
```

```
# Accessing values
sender_email = config_data['email_config']['sender_email']
sender_password = config_data['email_config']['sender_password']
reminder_message = config_data['reminder_message']
smtp_server_host = config_data['smtp_server']['host']
smtp_server_port = config_data['smtp_server']['port']

# Constants
DATABASE_HOST = config_data['mysql_data']['HOST']
DATABASE_USER = config_data['mysql_data']['USER']
DATABASE_PASSWORD = config_data['mysql_data']['PASSWORD']
DATABASE_NAME = config_data['mysql_data']['NAME']
def send_email(sender_email, sender_password, recipient_email, subject, body):
    # Compose the email message
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = recipient_email
    message["Subject"] = subject
    message.attach(MIMEText(body, "plain"))

    # Connect to the SMTP server and send the email
    with smtplib.SMTP(smtp_server_host, smtp_server_port) as server:
        server.starttls()
        server.login(sender_email, sender_password)
        server.sendmail(sender_email, recipient_email, message.as_string())

def send_hydration_reminder(username, email):

    # Set up your email configuration
    # sender_email = sender_email  # Replace with your email address
    # sender_password = sender_password  # Replace with your email password
    subject = "Hydration Reminder"

    # Compose the email body
    body = f"Hi {username},\n\nIt's time to drink water and stay hydrated! Remember your
daily goal.\n\nBest regards,\nYour Water Reminder App"

    # Send the email
    send_email(sender_email, sender_password, email, subject, body)

    print(f"Reminder for {username}: Hydration reminder email sent to {email}")

def schedule_reminders():
    try:
        # Create a connection to the MySQL database
        connection = mysql.connector.connect(
            host=DATABASE_HOST,
            user=DATABASE_USER,
            password=DATABASE_PASSWORD,
```

```
            database=DATABASE_NAME
        )

        cursor = connection.cursor()

        #     # Schedule reminders for each user
        cursor.execute('SELECT username, email FROM users')
        users = cursor.fetchall()

        # Schedule reminders for each user
        for user in users:
            username, email = user
            schedule.every(1).minutes.do(send_hydration_reminder, username=username,
email=email)
            schedule.every().day.at('08:00').do(send_hydration_reminder, username=username,
email=email)
            schedule.every().day.at('13:00').do(send_hydration_reminder, username=username,
email=email)
            schedule.every().day.at('17:00').do(send_hydration_reminder, username=username,
email=email)

        print("Reminder scheduled!")
    except mysql.connector.Error as err:
        print(f"Error: {err}")
    finally:
        connection.commit()
        connection.close()

def run_schedule():
    while True:
        schedule.run_pending()
        time.sleep(1)

def reminder_frame():
    # This function displays the reminder frame
    reminder_window = tk.Toplevel()
    reminder_window.title("Hydration Reminder")

    reminder_frame = ttk.Frame(reminder_window)
    reminder_frame.pack(padx=20, pady=20)

    reminder_message = ttk.Label(reminder_frame, text="It's time to drink water and stay
hydrated!")
    reminder_message.pack(pady=10)

    def deduct_goal():
        username = username_entry.get()  # Fetch the username entered
        user_profile = db.get_user_profile(username)
```

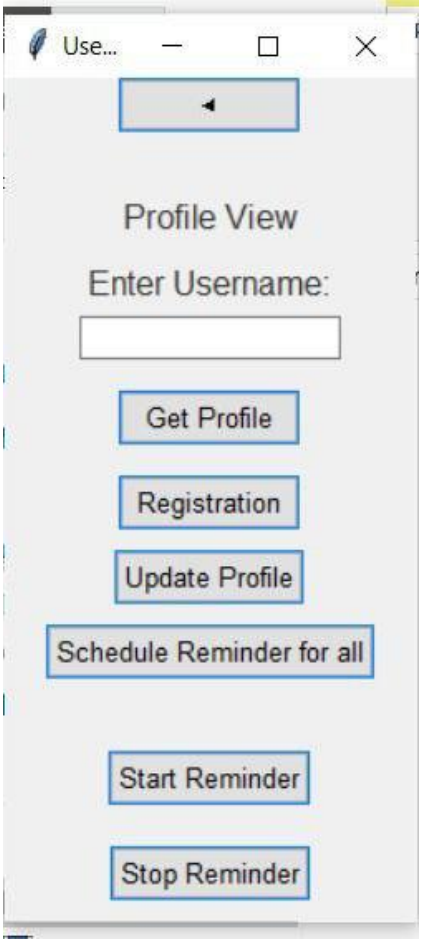
```
        if user_profile:
            # Assuming 'goal' is the field for the user's hydration goal in the database
            updated_goal = user_profile.get('goal', 0) - 1000  # Deducting 1 from the current goal

            # Update the user's goal in the database
            db.update_user_profile(username, new_goal=updated_goal)  # Update this line
            messagebox.showinfo("Goal Deducted", "Goal updated! Remember to stay
hydrated!")
        else:
            messagebox.showinfo("Update", "User not found!")

    done_button = ttk.Button(reminder_frame, text="Done", command=deduct_goal)
    done_button.pack(pady=10)



def start_reminder():
    global reminder_job
    # Schedule reminder to open reminder_frame function every 30 seconds
    reminder_job = schedule.every(30).seconds.do(reminder_frame)

def stop_reminder():
    global reminder_job
    # Cancel the scheduled reminder job
    if reminder_job:
        reminder_job.cancel()

def main_gui(username_entry_auth=None, password_entry_auth=None):
    global username_entry
    global password_entry
    global profile_frame
    global registration_frame
    global update_frame
    # ... (other global declarations)

    root = tk.Tk()
    root.title("User Profile Management")

    # Back Button (in registration and update frames)
    back_button = ttk.Button(
        text="\u25C0",  # Unicode for left arrow symbol
        command=lambda: switch_to_profile(),
        style="TButton"
    )
    back_button.pack()
    # Create frames for different screens
    profile_frame = tk.Frame(root)
    registration_frame = tk.Frame(root)
```

```
    update_frame = tk.Frame(root)

    # Set up widgets for registration screen
    # Styles
    style = ttk.Style()

    style.configure("TFrame", background="#f0f0f0")  # Set background color for frames
    style.configure("TLabel", background="#f0f0f0", foreground="#333", font=("Arial", 12))
# Label styles
    style.configure("TButton", background="#007bff", foreground="#000", font=("Arial",
10))  # Button styles

    # Create the registration frame
    registration_frame = ttk.Frame(root, style="TFrame")

    # Registration Labels
    labels = ["Username", "Email", "Password", "Goal", "Age", "Height", "Weight"]
    entries = []

    for label_text in labels:
        label = ttk.Label(registration_frame, text=f"Enter {label_text}:", style="TLabel")
        label.pack()
        entry = ttk.Entry(registration_frame)
        entry.pack(padx=10, pady=5)
        entries.append(entry)

    # Inside the registration frame setup
    register_button = ttk.Button(
        registration_frame,
        text="Register",
        command=lambda: register_user(*entries),
        style="TButton"
    )
    register_button.pack()

    # Create the update frame
    update_frame = ttk.Frame(root, style="TFrame")

    # Update Labels
    username_label_upd = ttk.Label(update_frame, text="Enter Username:", style="TLabel")
    username_label_upd.pack()
    username_entry_auth = ttk.Entry(update_frame)
    username_entry_auth.pack()

    password_label_upd = ttk.Label(update_frame, text="Enter Password:", style="TLabel")
    password_label_upd.pack()
    password_entry_auth = ttk.Entry(update_frame, show="*")
    password_entry_auth.pack()
```

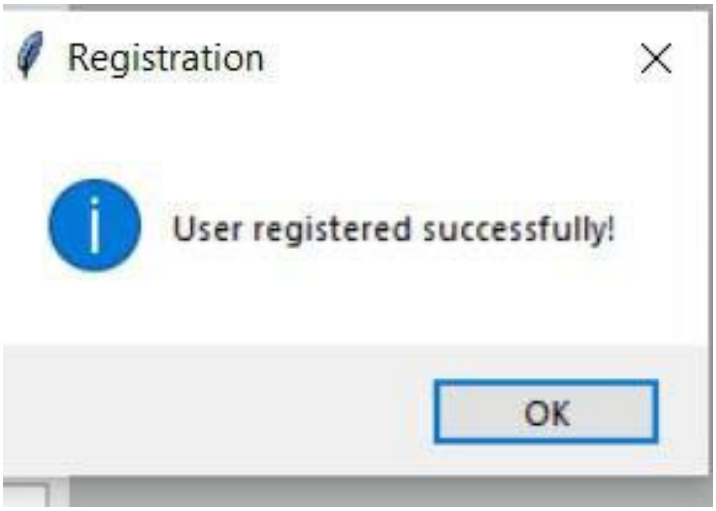
```
    # Update Button
    update_button = ttk.Button(
        update_frame,
        text="Update",
        command=lambda: authenticate_user(username_entry_auth, password_entry_auth),
        style="TButton"
    )
    update_button.pack()

    # Set up styles
    style = ttk.Style()
    style.configure("TFrame", background="#f0f0f0")  # Set background color for frames
    style.configure("TLabel", background="#f0f0f0", foreground="#333", font=("Arial", 12))
# Label styles
    style.configure("TButton", background="#007bff", foreground="#000", font=("Arial",
10))  # Button styles

    # Create the profile view frame
    profile_frame = ttk.Frame(root, style="TFrame")

    # Profile Label
    profile_label = ttk.Label(profile_frame, text="Profile View", style="TLabel")
    profile_label.pack(pady=10)  # Add padding around the label

    # Username Entry
    username_label_prof = ttk.Label(profile_frame, text="Enter Username:", style="TLabel")
    username_label_prof.pack()
    global username_entry
    username_entry = ttk.Entry(profile_frame)
    username_entry.pack(padx=10, pady=5)  # Add padding around the entry widget

    # Get Profile Button
    get_profile_button = ttk.Button(profile_frame, text="Get Profile", command=get_profile,
style="TButton")
    get_profile_button.pack(pady=10)  # Add padding around the button

    # Registration Button
    registration_button = ttk.Button(profile_frame, text="Registration",
command=switch_to_registration,
                                     style="TButton")
    registration_button.pack(pady=5)

    # Update Profile Button
    update_button = ttk.Button(profile_frame, text="Update Profile",
command=switch_to_update, style="TButton")
    update_button.pack(pady=5)

    profile_frame.pack(padx=20, pady=20)  # Add padding around the profile frame
```

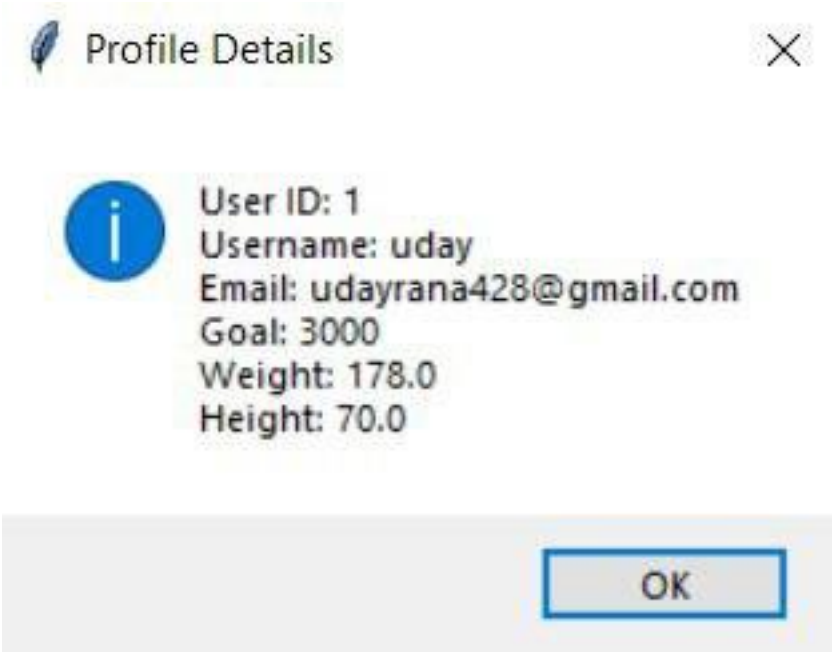
```
    # Schedule Reminder Button for all
    schedule_button = ttk.Button(profile_frame, text="Schedule Reminder for all",
                                 command=lambda: threading.Thread(target=schedule_reminders).start(),
style="TButton")
    schedule_button.pack(pady=5)

    # schedule reminder button for self

    start_reminder_button = ttk.Button(root, text="Start Reminder", command=start_reminder)
    start_reminder_button.pack(padx=20, pady=10)

    stop_reminder_button = ttk.Button(root, text="Stop Reminder", command=stop_reminder)
    stop_reminder_button.pack(padx=20, pady=10)

    # Start the scheduling loop in a separate thread
    threading.Thread(target=run_schedule).start()

    # Run the main loop for GUI
    root.mainloop()

if _name_ == "_main_":
    main_gui()
```

INPUT AND OUTPUT SCREEN




Start Reminder
Stop Reminder
Enter Username:
Uday1
Enter Email:
ranau@gmail.com
Enter Password:
Uday@123
Enter Goal:
3500
Enter Age:
22
Enter Height:
170
Enter Weight:
70
Register


Registration
User registered successfully!
OK


Profile Details
User ID: 1
Username: uday
Email: udayrana428@gmail.com
Goal: 3000
Weight: 178.0
Height: 70.0
OK

CONCLUSION

In conclusion, the script combines user interface design with database interactions and email functionalities to create a comprehensive user profile management system. Users can register, update their profiles, view information, and receive hydration reminders. The script showcases the versatility of Tkinter for GUI applications and incorporates threading for scheduling periodic reminders, enhancing user engagement and promoting a healthier lifestyle.