# FINAL CODE

#!/usr/bin/env python3  
import threading, time, os  
import RPi.GPIO as GPIO  
import mysql.connector  
import tkinter as tk  
from tkinter import messagebox  
from matplotlib.figure import Figure  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
import matplotlib.animation as animation  
  
# =====================  
# Settings and Globals  
# =====================  
# MariaDB settings  
DB\_HOST = "localhost"  
DB\_USER = "team5"  
DB\_PASSWORD = "1234"  
DB\_NAME = "mydb"  
  
# Flow sensor (DYF-DN40) settings  
FLOW\_SENSOR\_PIN = 18            # GPIO pin for pulse output  
CALIBRATION\_FACTOR = 0.45       # Calibration factor (pulses per liter)  
PIPE\_AREA = 0.001284            # Pipe cross-sectional area (m²)  
  
# DS18B20 temperature sensor paths  
# Mapping: temp1 is sensor with ID 28-77e7d445439d (Temp Sensor 1),  
#          temp2 is sensor with ID 28-2df8d4463b04 (Temp Sensor 2)  
SENSOR\_PATHS = {  
    "temp1": "/sys/bus/w1/devices/28-77e7d445439d/w1\_slave",  
    "temp2": "/sys/bus/w1/devices/28-2df8d4463b04/w1\_slave"  
}  
  
# Global data arrays for plotting  
time\_data = []  
temp1\_data = []  
temp2\_data = []  
flow\_rate\_data = []  
flow\_velocity\_data = []  
  
start\_time = time.time()  
monitoring = False  
data\_lock = threading.Lock()  
monitor\_start\_time = None  
  
# =====================  
# GPIO Initialization  
# =====================  
GPIO.setmode(GPIO.BCM)  
GPIO.setup(FLOW\_SENSOR\_PIN, [GPIO.IN](http://gpio.in/), pull\_up\_down=GPIO.PUD\_UP)  
  
# =====================  
# Sensor reading functions  
# =====================  
def read\_temperature(sensor\_path):  
    try:  
        with open(sensor\_path, "r") as f:  
            lines = f.readlines()  
        # Wait until first line ends with "YES"  
        while lines[0].strip()[-3:] != "YES":  
            time.sleep(0.1)  
            with open(sensor\_path, "r") as f:  
                lines = f.readlines()  
        temp\_str = lines[1].split("t=")[-1]  
        return float(temp\_str) / 1000.0  # Celsius  
    except Exception as e:  
        print(f"Temperature sensor error ({sensor\_path}): {e}")  
        return None  
  
def measure\_flow():  
    """Measure flow sensor pulses for 1 second."""  
    pulse\_count = 0  
    t\_start = time.time()  
    duration = 1.0  
    while time.time() - t\_start < duration:  
        try:  
            if GPIO.input(FLOW\_SENSOR\_PIN) == 0:  
                pulse\_count += 1  
                # 내부 while 루프에서도 종료 플래그 확인  
                while GPIO.input(FLOW\_SENSOR\_PIN) == 0:  
                    if not monitoring:  
                        break  
        except RuntimeError as e:  
            # GPIO가 cleanup 되었거나 호출에 문제가 있을 경우, 예외를 잡아서 기본값 반환  
            print(f"RuntimeError in measure\_flow: {e}")  
            return 0.0, 0.0  
        if not monitoring:  
            break  
    if pulse\_count == 0:  
        return 0.0, 0.0  
    flow\_rate = (pulse\_count / CALIBRATION\_FACTOR)  # L/min  
    flow\_rate\_m3s = (flow\_rate \* 1e-3) / 60           # Convert L/min to m³/s  
    flow\_velocity = flow\_rate\_m3s / PIPE\_AREA           # m/s  
    return flow\_rate, flow\_velocity  
  
def sensor\_monitoring():  
    global monitoring, monitor\_start\_time  
    monitor\_start\_time = time.time()  
    while monitoring:  
        try:  
            flow\_rate, flow\_velocity = measure\_flow()  
        except Exception as e:  
            print(f"Exception in sensor\_monitoring during measure\_flow: {e}")  
            flow\_rate, flow\_velocity = 0.0, 0.0  
  
        t1 = read\_temperature(SENSOR\_PATHS["temp1"])  
        t2 = read\_temperature(SENSOR\_PATHS["temp2"])  
        temperatures = {"temp1": t1, "temp2": t2}  
        insert\_data(flow\_rate, flow\_velocity, temperatures)  
        current\_time = time.time() - monitor\_start\_time  
        with data\_lock:  
            time\_data.append(current\_time)  
            temp1\_data.append(t1)  
            temp2\_data.append(t2)  
            flow\_rate\_data.append(flow\_rate)  
            flow\_velocity\_data.append(flow\_velocity)  
            if current\_time > 60:  
                time\_data.pop(0)  
                temp1\_data.pop(0)  
                temp2\_data.pop(0)  
                flow\_rate\_data.pop(0)  
                flow\_velocity\_data.pop(0)  
        print(f"Monitoring: {current\_time:.1f}s, flow\_rate={flow\_rate:.2f}, flow\_velocity={flow\_velocity:.3f}, temp1={t1}, temp2={t2}")  
        time.sleep(3)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      
  
def insert\_data(flow\_rate, flow\_velocity, temperatures):  
    try:  
        conn = mysql.connector.connect(  
            host=DB\_HOST,  
            user=DB\_USER,  
            password=DB\_PASSWORD,  
            database=DB\_NAME  
        )  
        cursor = conn.cursor()  
        sql = """  
        INSERT INTO sensor\_data (flow\_rate, flow\_velocity, temp1, temp2, timestamp)  
        VALUES (%s, %s, %s, %s, NOW())  
        """  
        cursor.execute(sql, (flow\_rate, flow\_velocity, temperatures["temp1"], temperatures["temp2"]))  
        conn.commit()  
        cursor.close()  
        conn.close()  
        print(f"Data saved: flow\_rate={flow\_rate:.2f} L/min, flow\_velocity={flow\_velocity:.3f} m/s, temperatures={temperatures}")  
    except Exception as e:  
        print(f"Database error: {e}")  
  
  
# =====================  
# GUI: Login Window (Modal)  
# =====================  
class LoginWindow(tk.Toplevel):  
    def \_\_init\_\_(self, master):  
        super().\_\_init\_\_(master)  
        self.title("Login")  
        self.geometry("300x150")  
        self.resizable(False, False)  
        self.update\_idletasks()  
        screen\_width = self.winfo\_screenwidth()  
        screen\_height = self.winfo\_screenheight()  
        x = (screen\_width - 300) // 2  
        y = (screen\_height - 150) // 2  
        self.geometry(f"300x150+{x}+{y}")  
        tk.Label(self, text="Username:").pack(pady=5)  
        self.entry\_user = tk.Entry(self)  
        self.entry\_user.pack()  
        tk.Label(self, text="Password:").pack(pady=5)  
        self.entry\_pass = tk.Entry(self, show="\*")  
        self.entry\_pass.pack()  
        self.btn\_login = tk.Button(self, text="Login", command=self.check\_login)  
        self.btn\_login.pack(pady=10)  
        self.protocol("WM\_DELETE\_WINDOW", self.on\_close)  
        self.success = False  
  
    def check\_login(self):  
        username = self.entry\_user.get()  
        password = self.entry\_pass.get()  
        if username == "admin" and password == "password":  
            self.success = True  
            self.destroy()  
        else:  
            messagebox.showerror("Login Failed", "Incorrect username or password.")  
  
    def on\_close(self):  
        self.master.destroy()  
  
# =====================  
# GUI: Main Application Window  
# =====================  
class MainApp(tk.Tk):  
    def \_\_init\_\_(self):  
        super().\_\_init\_\_()  
        self.title("RMS - Sensor Monitoring")  
        self.geometry("1000x800")  
        self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)  
        # Control frame with buttons  
        control\_frame = tk.Frame(self)  
        control\_frame.pack(side=tk.TOP, fill=tk.X)  
        self.btn\_start = tk.Button(control\_frame, text="Monitor Start", command=self.start\_monitoring)  
        self.btn\_start.pack(side=tk.LEFT, padx=10, pady=5)  
        self.btn\_stop = tk.Button(control\_frame, text="Stop", command=self.stop\_monitoring, state=tk.DISABLED)  
        self.btn\_stop.pack(side=tk.LEFT, padx=10, pady=5)  
        self.btn\_end = tk.Button(control\_frame, text="End", command=self.on\_closing)  
        self.btn\_end.pack(side=tk.LEFT, padx=10, pady=5)  
  
        # Create matplotlib figure and adjust spacing  
        self.fig = Figure(figsize=(10, 8))  
        self.fig.subplots\_adjust(hspace=0.6, wspace=0.4)  
        # Row 1: Temperature line graph  
        self.ax\_temp\_line = self.fig.add\_subplot(321)  
        self.line\_temp1, = self.ax\_temp\_line.plot([], [], label="Temp Sensor 1", color="red")  
        self.line\_temp2, = self.ax\_temp\_line.plot([], [], label="Temp Sensor 2", color="blue")  
        self.ax\_temp\_line.set\_title("Temperature Data")  
        self.ax\_temp\_line.set\_xlabel("Time (s)")  
        self.ax\_temp\_line.set\_ylabel("Temperature (°C)")  
        self.ax\_temp\_line.legend()  
        # Row 2, Column 1: Flow Rate graph  
        self.ax\_flow\_rate = self.fig.add\_subplot(322)  
        self.line\_flow\_rate, = self.ax\_flow\_rate.plot([], [], label="Flow Rate", color="green")  
        self.ax\_flow\_rate.set\_title("Flow Rate (L/min)")  
        self.ax\_flow\_rate.set\_xlabel("Time (s)")  
        self.ax\_flow\_rate.set\_ylabel("L/min")  
        self.ax\_flow\_rate.legend()  
        # Row 2, Column 2: Flow Velocity graph  
        self.ax\_flow\_velocity = self.fig.add\_subplot(323)  
        self.line\_flow\_velocity, = self.ax\_flow\_velocity.plot([], [], label="Flow Velocity", color="purple")  
        self.ax\_flow\_velocity.set\_title("Flow Velocity (m/s)")  
        self.ax\_flow\_velocity.set\_xlabel("Time (s)")  
        self.ax\_flow\_velocity.set\_ylabel("m/s")  
        self.ax\_flow\_velocity.legend()  
        # Row 3, Column 1: Bar gauge for Temp Sensor 2 (left)  
        self.ax\_gauge\_temp2 = self.fig.add\_subplot(324)  
        # Row 3, Column 2: Bar gauge for Temp Sensor 1 (right)  
        self.ax\_gauge\_temp1 = self.fig.add\_subplot(325)  
        self.ax\_gauge\_temp2.set\_xlim(0, 100)  
        self.ax\_gauge\_temp2.set\_ylim(0, 1)  
        self.ax\_gauge\_temp2.get\_yaxis().set\_visible(False)  
        self.ax\_gauge\_temp1.set\_xlim(0, 100)  
        self.ax\_gauge\_temp1.set\_ylim(0, 1)  
        self.ax\_gauge\_temp1.get\_yaxis().set\_visible(False)  
  
        self.canvas = FigureCanvasTkAgg(self.fig, master=self)  
        self.canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)  
  
        self.monitor\_thread = None  
        self.plot\_after\_id = None  
        self.update\_plot()  
  
    def start\_monitoring(self):  
        global monitoring  
        if not monitoring:  
            monitoring = True  
            self.btn\_start.config(state=tk.DISABLED)  
            self.btn\_stop.config(state=tk.NORMAL)  
            self.monitor\_thread = threading.Thread(target=sensor\_monitoring, daemon=True)  
            self.monitor\_thread.start()  
  
    def stop\_monitoring(self):  
        global monitoring  
        monitoring = False  
        self.btn\_start.config(state=tk.NORMAL)  
        self.btn\_stop.config(state=tk.DISABLED)  
  
    def on\_closing(self):  
        global monitoring  
        monitoring = False  
        try:  
            if self.monitor\_thread is not None:  
                self.monitor\_thread.join(timeout=2)  
        except Exception:  
            pass  
        try:  
            if self.plot\_after\_id is not None:  
                self.after\_cancel(self.plot\_after\_id)  
        except Exception:  
            pass  
        self.destroy()  
  
    def update\_plot(self):  
    # 창이 존재하지 않으면 업데이트 중지  
        if not self.winfo\_exists():  
            return  
  
        with data\_lock:  
            t = list(time\_data)  
            t1 = list(temp1\_data)  
            t2 = list(temp2\_data)  
            fr = list(flow\_rate\_data)  
            fv = list(flow\_velocity\_data)  
        self.line\_temp1.set\_data(t, t1)  
        self.line\_temp2.set\_data(t, t2)  
        self.ax\_temp\_line.relim()  
        self.ax\_temp\_line.autoscale\_view()  
        self.line\_flow\_rate.set\_data(t, fr)  
        self.ax\_flow\_rate.relim()  
        self.ax\_flow\_rate.autoscale\_view()  
        self.line\_flow\_velocity.set\_data(t, fv)  
        self.ax\_flow\_velocity.relim()  
        self.ax\_flow\_velocity.autoscale\_view()  
  
    # 업데이트 바 게이지  
        self.ax\_gauge\_temp2.cla()  
        self.ax\_gauge\_temp2.set\_xlim(0, 100)  
        self.ax\_gauge\_temp2.set\_ylim(0, 1)  
        self.ax\_gauge\_temp2.get\_yaxis().set\_visible(False)  
        current\_temp2 = t2[-1] if t2 and t2[-1] is not None else 0  
        self.ax\_gauge\_temp2.barh(0.5, current\_temp2, height=0.5, color="blue")  
        self.ax\_gauge\_temp2.set\_title("Temp Sensor 2")  
        self.ax\_gauge\_temp2.text(50, 0.5, f"{current\_temp2:.1f}°C", ha="center", va="center", color="white", fontsize=12)  
  
        self.ax\_gauge\_temp1.cla()  
        self.ax\_gauge\_temp1.set\_xlim(0, 100)  
        self.ax\_gauge\_temp1.set\_ylim(0, 1)  
        self.ax\_gauge\_temp1.get\_yaxis().set\_visible(False)  
        current\_temp1 = t1[-1] if t1 and t1[-1] is not None else 0  
        self.ax\_gauge\_temp1.barh(0.5, current\_temp1, height=0.5, color="red")  
        self.ax\_gauge\_temp1.set\_title("Temp Sensor 1")  
        self.ax\_gauge\_temp1.text(50, 0.5, f"{current\_temp1:.1f}°C", ha="center", va="center", color="white", fontsize=12)  
  
        self.canvas.draw()  
        # 예약된 콜백을 다시 등록 (창이 존재할 경우에만)  
        try:  
            self.plot\_after\_id = self.after(4000, self.update\_plot)  
        except tk.TclError:  
            pass  
  
# =====================  
# Main execution  
# =====================  
if \_\_name\_\_ == "\_\_main\_\_":  
    from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
    temp\_root = tk.Tk()  
    temp\_root.withdraw()  
    login = LoginWindow(temp\_root)  
    temp\_root.wait\_window(login)  
    if not getattr(login, "success", False):  
        temp\_root.destroy()  
        exit()  
    temp\_root.destroy()  
    app = MainApp()  
    app.mainloop()  
    GPIO.cleanup()