Gmail   **pp Pettina <pppettina@gmail.com>**

code

1 message

pp Pettina <pppettina@gmail.com>   Tue, Apr 15, 2025 at 2:13 PM
To: pp Pettina <pppettina@gmail.com>

```
from flask import Flask, render_template, jsonify, request
from flask_socketio import SocketIO
from sense_emu import SenseHat
import sqlite3
import time
from datetime import datetime
import os

app = Flask(__name__)
app.secret_key = os.urandom(16)
socketio = SocketIO(app)
sense = SenseHat()
DB_PATH = 'environment.db'

# === Added calibration functions ===
def read_raw_sensor_values():
    """Read raw 16-bit unsigned integers from sensor registers"""
    try:
        # Access raw register values
        temp_raw = sense._temperature  # Temperature register (16-bit)
        humi_raw = sense._humidity     # Humidity register (16-bit)
        return temp_raw, humi_raw
    except AttributeError as e:
        print(f"Sensor register error: {str(e)}")
        return 0, 0  # Return safe defaults

def calibrate_values(temp_raw, humi_raw):
    """Convert raw sensor values to actual measurements"""
    try:
        # Conversion formulas based on sensor datasheet
        temperature = -45 + 175 * (temp_raw / 65535.0)
        humidity = 100 * (humi_raw / 65535.0)

        # Apply environmental compensation (adjust these values as needed)
        temperature -= 2.5   # Temperature offset calibration
        humidity *= 0.9      # Humidity scaling factor

        return round(temperature, 1), round(humidity, 1)
    except Exception as e:
        print(f"Calibration error: {str(e)}")
        return 0.0, 0.0  # Return safe values
# === End of calibration functions ===

def get_db():
    return sqlite3.connect(DB_PATH)

def data_loop():
    while True:
        try:
            # === Modified data collection ===
            # Get raw sensor values
            temp_raw, humi_raw = read_raw_sensor_values()

            # Apply calibration
            temp, humidity = calibrate_values(temp_raw, humi_raw)

            # Pressure sensor typically works correctly
            pressure = round(sense.get_pressure(), 1)
            # === End of modified section ===

            timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

            # Debug output
            print(f"[Valid Data] Temp: {temp}C, Humidity: {humidity}%, Pressure: {pressure}hPa")

            # Database operations
            with get_db() as conn:
                # Store readings
                conn.execute('''
                    INSERT INTO sensor_data (timestamp, temp, humidity, pressure)
                    VALUES (?, ?, ?, ?)
                ''', (timestamp, temp, humidity, pressure))
                conn.commit()

                # Check thresholds
                thresholds = conn.execute('SELECT * FROM thresholds').fetchone()
                alerts = {
                    'temp': temp < thresholds[1] or temp > thresholds[2],
                    'humi': humidity < thresholds[3] or humidity > thresholds[4],
                    'pres': pressure < thresholds[5] or pressure > thresholds[6]
                }

            # Send update to frontend
            socketio.emit('update', {
                'time': timestamp,
                'temp': temp,
                'humidity': humidity,
                'pressure': pressure,
                'alerts': alerts
            })

            time.sleep(10)

        except Exception as e:
            print(f"Data collection error: {str(e)}")
            time.sleep(30)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/history')
def get_history():
    with get_db() as conn:
        data = conn.execute('''
            SELECT timestamp, temp
            FROM sensor_data
            ORDER BY timestamp DESC
            LIMIT 50
        ''').fetchall()
    return jsonify([{'time': d[0], 'temp': d[1]} for d in data])

@app.route('/thresholds', methods=['GET', 'POST'])
def handle_thresholds():
    with get_db() as conn:
        if request.method == 'POST':
            new = request.json
            conn.execute('''
                UPDATE thresholds SET
                min_temp = ?, max_temp = ?,
                min_humi = ?, max_humi = ?,
                min_pres = ?, max_pres = ?
            ''', tuple(new.values()))
            conn.commit()
            return jsonify({'status': 'success'})
        else:
```

```
        return jsonify(dict(zip(
            ['min_temp','max_temp','min_humi','max_humi','min_pres','max_pres'],
            conn.execute('SELECT * FROM thresholds').fetchone()[1:]
        )))

if __name__ == '__main__':
    import threading
    threading.Thread(target=data_loop, daemon=True).start()
    socketio.run(app, host='0.0.0.0', port=5000)
```