PART 1 IMPLEMENTATION/ASSUMPTIONS

Implementation details

The project has been implemented using Python, html, JavaScript, the Tornado server and SQLite. Sensor sampling is implemented by the pseudo code provided by the class. All acquisitions are stored in the database and is persistent on disk. Listed below are the broad details of the implemented code.

Note that I could have used the Tornado template for the client instead of html/JavaScipt but decided to write the scripts in JavaScript using WebSocket.

Create an SQLite Database to store humidity/temperature readings

1. The schema for the database is: id (INTEGER), timestamp (TEXT, seconds since epoch), humidity (REAL, percentage), temperature (REAL, Fahrenheit).
2. Create methods:
    a. Create database and table if it does not exist.
    b. Operations to count samples, insert, query and produce avg, min, max statistics.

Create the Tornado Server

Create a Tornado server:

1. Start server and prepare to listen and process requests.
2. If the database does not exist, create it.
3. The server listens to clients requesting sensor operations:
    a. Read and log a sample into the database – Message to client.
    b. Query statistics for the last n samples – Message them to client.

Create the html User Interface

The html page consists of:

1. A button to read a single reading.
2. A button to read 10 samples.
3. A button to show statistics on the last 10 or fewer readings, including minimum, maximum and average in the units set by the user. Samples will be graphed using chart.js.
4. Spin boxes to enable the user to set alarm values for humidity and temperature. Valid values are between 0-100% and -20-100 degrees Fahrenheit.
5. A text widget to show values as they are acquired, queried.
6. A button to shut down the clients.
7. Events are handled by JavaScript and the sockets connected to the Tornado server.

PART 2 – THE CODE

Messages follow the json format:

If the Input to the server is:

```
{
        "action": "read"
}
```

The response to the client will be:

```
{
        "response": "reading",
        "humidity": humidity,
        "temperature": temperature
}
```

If the input to the server is:

```
{
        "action": "stats",
        "samples": 10
}
```

The response to the client will be:

```
{
        "response": "stats",
        "humidity-min": h-min,
        "humidity-max": h-max,
        "humidity-avg": h-avg,
        "temperature-min": t-min,
        "temperature-max": t-max,
        "temperarure-avg": t-avg,
        "humidity-array": [t1, t10]
        "temperature-array": [t1, t10]


}
```

sensorserver.py

```python
import tornado.ioloop
import tornado.web
import tornado.websocket
import nest_asyncio
import logging
import json
import datetime
from pseudoSensor import PseudoSensor
from humtempdb import HumTempDB

class Application(tornado.web.Application):
        def __init__(self,db,ps):
                logging.basicConfig(filename='sensorserver.log', filemode='w', le
vel=logging.DEBUG)
                handlers = [(r"/", MainHandler),
                (r"/ws", WSHandler),
                (r'/css/(.*)', tornado.web.StaticFileHandler, {'path': './css'}),
                (r'/scripts/(.*)', tornado.web.StaticFileHandler, {'path': './scr
ipts'}),
                ]
                settings = dict(debug=True)
                tornado.web.Application.__init__(self, handlers, settings)
                self.ps = ps
                self.db = db

class MainHandler(tornado.web.RequestHandler) :
    def get(self):
            self.render("sensorUI.html")

class WSHandler(tornado.websocket.WebSocketHandler):
        def open(self):
                logging.info("Connection to websocket open")

        def on_close(self):
                logging.info("A client disconnected")

        def on_message(self, message):
                logging.info("message: {}".format(message))
                jsonObject = json.loads(message)
                action = jsonObject["action"]
                response = {}
                if action == "single":
                        ts, hr, tr = self.get_1sample()
                        response["timestamp"] = ts
                        response["response"] = "reading"
                        response["humidity"] = hr
                        response["temperature"] = tr
                elif action == "stats":
                        rows, stats = self.get_stats()
                        response["response"] = "stats"
                        response["humidity-min"] = stats[1]
                        response["humidity-max"] = stats[2]
                        response["humidity-avg"] = stats[0]
                        response["temperature-min"] = stats[4]
                        response["temperature-max"] = stats[5]
                        response["temperature-avg"] = stats[3]
                        dataHum = []
                        dataTemp = []
                        for sample in rows:
                                dataHum.append(sample[2])
```

```python
                                dataTemp.append(sample[3])
                        response["humidity-array"] = dataHum
                        response["temperature-array"] = dataTemp
                self.write_message(json.dumps(response))

        def get_1sample(self):
                logging.info("Getting 1 sample")
                h,t = self.application.ps.generate_values()
                hr = round(h)
                tr = round(t)
                ts = get_timestamp()
                self.application.db.insert_record(ts,hr,tr)
                return ts,hr,tr

        def get_stats(self):
                rows = self.application.db.query_samples(10)
                stats = self.application.db.stats_samples(10)
                return rows, stats

def get_timestamp():
        ct = datetime.datetime.now()
        ts = ct.timestamp()
        return ts


if __name__ == "__main__":
    nest_asyncio.apply()
    database = r"C:\Users\laura\work\db\humtemp.db"
    db = HumTempDB(database)
    ps = PseudoSensor()
    app = Application(db,ps)
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

sensorUI.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Sensor Data Manager</title>
        <link rel="stylesheet" href="css/style.css" type="text/css">
        <script src="scripts/script.js" type="text/javascript" defer></script>
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
        <header>
                <h1>Sensor Data Manager</h1>
        </header>

        <main>
                <div class="container1">

                        <a href="#" class="button" id="singlereq">
                                Request one humidity-temperature reading
                        </a>

                        <a href="#" class="button" id="multireq">
                                Request 10 humidity-temperature readings
                        </a>

                        <a href="#" class="button" id="statsreq">
                                Request stats for last 10 samples
                        </a>
                </div>

                <br>
                <br>

                <div class="container2">

                        <div>
                                <label class="lb" for="hum"><b>Alarm for humidity
:</b></label>
                                <input type="number" name="hum" class="spinb" id=
"spinh" min="0" max="100" step="1" value="50"  size=6>
                        </div>


                        <div>
                                <label class="lb" for="temp"><b>Alarm for tempera
ture:</b></label>
                                <input type="number" name="temp" class="spinb" id
="spint" min="-20" max="100" step="1" value="50"  size=6>
                        </div>

                </div>
                <br>
                <br>

                <div class="container1">

                        <a href="#" class="button" id="closeb">
                                Close down
                        </a>

                </div>

                <div>
                        <div style="width:50%; float:left" id="results">

                        </div>

                        <div style="width:50%; float:right" id="graph">
                                <canvas id="myChart"></canvas>
                        </div>
                </div>
        </main>

</body>

</html>
```

style.css

```css
header {
        padding: 50px;
        background-color: lightblue;
        text-align: center;
        color: black;
        margin-bottom: 20px;
 }

.button {
        background-color: blue;
        width: 400px;
        border: 2px solid black;
        color: white;
        padding: 15px 32px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 18px;
        margin-right: 20px;
}

 .container1 {
         border: 2x solid black;
         border-radius: 2px;
         display: flex;
         justify-content: flex-start;
 }

 .container2{
         border: 2x solid black;
         border-radius: 2px;
         display: flexi-column;
         justify-content: space-around;
 }

 .spinb {
         width: 30px;
 }

 label {
        display: block;
        margin-bottom: .5em;
}

 .lb {
         white-space: pre;
 }
```

script.css

```
(function(){

        "use strict";

        // Websocket address
        const myWebSocket = "ws://localhost:8888/ws";

// Websocket variable
        var socket = new WebSocket(myWebSocket);
        socket.onerror = socketErrorAlert;
```

```
// Websocket event handlers
        if (socket){
                socket.onopen = function() {
                        console.log("Server is ready");
                }
                socket.onmessage = function(msg){
                        console.log("Got message from server - calling processRes
ponse");
                        processResponse(msg);
                }
                socket.onclose = function() {
                        console.log("Server connection closed");
                }
        }
        else {
                console.log("Invalid websocket");
        }

// Json requests
        var jsonSingleString = '{"action":"single"}';
        var jsonStatsString = '{"action":"stats", "samples":10}';

// Misc variables
        const spinBoxH = document.getElementById("spinh");
        const spinBoxT = document.getElementById("spint");
        var temp = 0;
        var hum = 0;
        var tAlarm = 100;
        var hAlarm = 100;
        var hMin, hMax, hAvg, tMin, tMax, tAvg;
        var myChart;

// Button handlers
        const singleRequestButton = document.getElementById("singlereq");
        const multiRequestButton = document.getElementById("multireq");
        const statsButton = document.getElementById("statsreq");
        const closeButton = document.getElementById("closeb");

        singleRequestButton.addEventListener("click", function(event){
                read1();
        });

        multiRequestButton.addEventListener("click", function(event){
                read1();
                for(let i=0; i<9; i++){
                        setTimeout(read1,1000);
                }
        });

        statsButton.addEventListener("click", function(event){
                socket.send(jsonStatsString);
        });

        closeButton.addEventListener("click", function(event){
                socket.close();
                alert("Server connection closed - Close tab manually");
        });


        function socketErrorAlert(event){
                alert("Server/socket error: " + event.data);
        }


        function read1(){
                if (socket.readyState !== WebSocket.CLOSED && socket.readyState !
== WebSocket.CLOSING) {
                        socket.send(jsonSingleString);
                }
                else {
                        alert("Server or socket not available");
                }
        }

        function processResponse(jsonText){
                console.log(jsonText);
                const jsonObject = JSON.parse(jsonText.data);
                const response = jsonObject["response"];
                if (response == "reading"){
                        const ts = jsonObject["timestamp"];
                        const hum = jsonObject["humidity"];
                        const temp = jsonObject["temperature"];
                        const text1 = `+++ At timestamp ${ts} Humidity: ${hum} %
+++ Temperature: ${temp} Fahrenheit +++`;
                        showResults(text1);
                        const hAlarm = spinBoxH.value;
                        const tAlarm = spinBoxT.value;
                        if (hum > hAlarm) {
                                showResults("<b>Alarm: humidity level exceeded!</
b>");
```

```
            }
            if (temp > tAlarm) {
                showResults("<b>Alarm: temperature level exceeded!</b>");
            }
            console.log(`Got Humidity ${hum} Temperature ${temp}`);
        }
        else if (response == "stats"){
            console.log("Processing sensor stats");
            const hMin = jsonObject["humidity-min"];
            const hMax = jsonObject["humidity-max"];
            const hAvg = jsonObject["humidity-avg"];
            const tMin = jsonObject["temperature-min"];
            const tMax = jsonObject["temperature-max"];
            const tAvg = jsonObject["temperature-avg"];
            var text2 = `+++ Minimum Humidity ${hMin} +++ Maximum Humidity ${hMax} +++ Average Humidity ${hAvg}`;
            showResults(text2);
            text2 = `+++ Minimum Temperature ${tMin} +++ Maximum Temperature ${tMax} +++ Average Temperature ${tAvg}`;
            showResults(text2);
            var dataHum = jsonObject["humidity-array"];
            var dataTemp = jsonObject["temperature-array"];
            showGraph(dataHum,dataTemp);
        }
        else {
            console.log("Unknown response");
        }

    }

    function showResults(texto){
        var par = document.createElement('p');
        par.innerHTML = texto;
        document.getElementById("results").appendChild(par);
    }

    function showGraph(dataHum,dataTemp) {
//Graph configuration and setup

        console.log("Processing graph");

        var labls=[];
        for(let i=1;i<dataTemp.length+1;i++){
            labls.push(i);
```

```
                }

                const data = {
                  labels: labls,
                  datasets: [{
                        label: 'humidity',
                        backgroundColor: 'rgb(0, 255, 0)',
                        borderColor: 'rgb(0, 255, 0)',
                        data: dataHum,
                  },
                  {
                        label: 'temperature',
                        backgroundColor: 'rgb(255, 0, 0)',
                        borderColor: 'rgb(255, 0, 0)',
                        data: dataTemp,

                  }]
                };

                const config = {
                        type: 'line',
                        data,
                        options: {}
                };


                if (myChart) {
                        myChart.destroy();
                }
                myChart = new Chart(document.getElementById('myChart'),config);
        }

})();
```

humtempdb.py

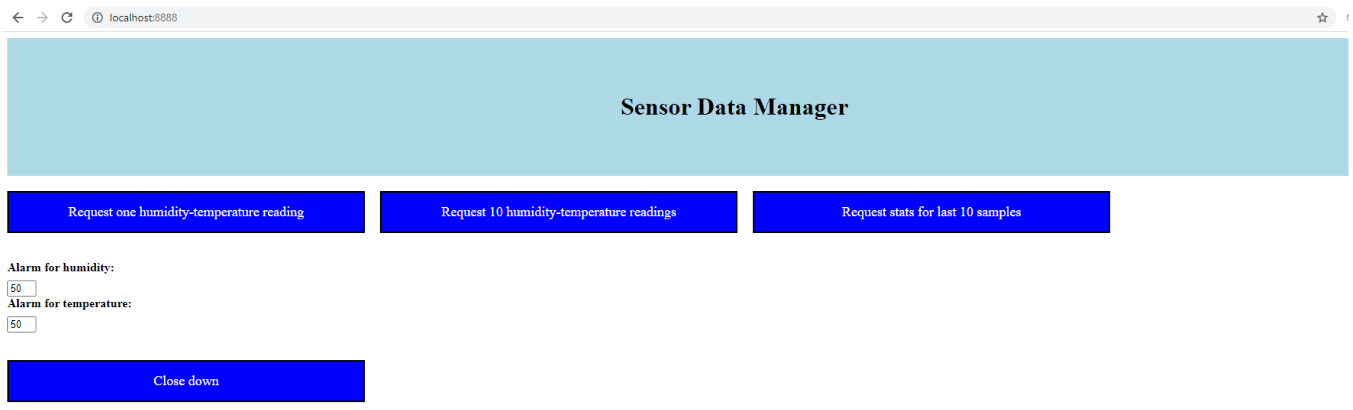
```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  2 18:07:51 2021

@author: laura
"""
```

```python
import sqlite3
from sqlite3 import Error

class HumTempDB:
    def __init__(self,database):
        self.database = database
        self.conn = None
        self.lastId = 0
        self.setup_table()

    def setup_table(self):
        """Setup the database/table"""
        sql_create_samples_table = """CREATE TABLE IF NOT EXISTS samples (
                                        id integer PRIMARY KEY,
                                        timestamp text,
                                        humidity integer,
                                        temperature integer); """
        # create a database connection
        self.conn = self.create_connection()
        # create tables
        if self.conn is not None:
            # create samples table
            self.create_table(sql_create_samples_table)
            self.lastId = self.total_samples()
        else:
            print("Error! cannot create the database connection.")

    def create_connection(self):
        """ create a database connection to a SQLite database """
        conn = None
        try:
            conn = sqlite3.connect(self.database)
            print(sqlite3.version)
        except Error as e:
            print(e)
        return conn

    def create_table(self,create_table_sql):
        """ create a table from the create_table_sql statement
        :param conn: Connection object
        :param create_table_sql: a CREATE TABLE statement
        :return:
        """
        try:
            c = self.conn.cursor()
            c.execute(create_table_sql)
        except Error as e:
            print(e)


    def insert_record(self,timestamp, humidity, temperature):
        """ Insert an id, timestamp, humidity, temperature record"""
        self.lastId  = self.lastId + 1

        sql = """INSERT INTO samples(id,timestamp,humidity,temperature)
                VALUES(?, ?, ?, ?) """
        cur = self.conn.cursor()
        cur.execute(sql,(self.lastId,timestamp,humidity,temperature))
        self.conn.commit()
        return cur.lastrowid

    def total_samples(self):
        """Count the # of samples stored in the database"""
        cur = self.conn.cursor()
        cur.execute("SELECT * FROM samples")
        results = cur.fetchall()
        return len(results)

    def query_samples(self,lastn):
        """Retrieve the last n samples"""
        cur = self.conn.cursor()
        idmin = self.lastId - lastn
        cur.execute("SELECT * FROM samples WHERE id > ? ", (idmin,))
        rows = cur.fetchall()
        return rows

    def stats_samples(self,lastn):
        """Retrieve stats for the last n samples"""
        cur = self.conn.cursor()
        sql = \
        """SELECT avg(humidity),min(humidity), max(humidity), avg(temperature),
        min(temperature), max(temperature)
        FROM samples
        WHERE id > ? """

        idmin = self.lastId - lastn
        findAll = cur.execute(sql,(idmin,))
        all = findAll.fetchone()
        print(all)
        return all

    def shutdown(self):
        """Shutdown database"""
        self.conn.commit()
        self.conn.close()

if __name__ == '__main__':
    database = r"C:\Users\laura\work\db\humtemp.db"
    db = HumTempDB(database)
```

pseudoSensor.py

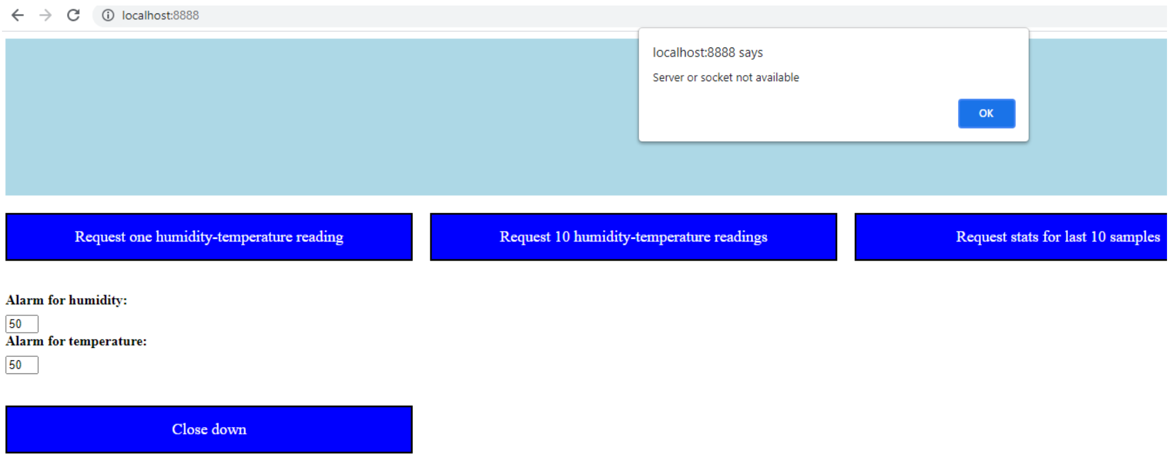
```
import random

class PseudoSensor:

    h_range = [0, 20, 20, 40, 40, 60, 60, 80, 80, 90, 70, 70, 50, 50, 30, 30, 10,
 10]

    t_range = [-20, -10, 0, 10, 30, 50, 70, 80, 90, 80, 60, 40, 20, 10, 0, -10]

    h_range_index = 0

    t_range_index = 0

    humVal = 0

    tempVal = 0


    def __init__(self):

        self.humVal = self.h_range[self.h_range_index]

        self.tempVal = self.t_range[self.t_range_index]



    def generate_values(self):

        self.humVal = self.h_range[self.h_range_index] + random.uniform(0, 10);
        self.tempVal = self.t_range[self.t_range_index] + random.uniform(0, 10);

        self.h_range_index += 1

        if self.h_range_index > len(self.h_range) - 1:

            self.h_range_index = 0

        self.t_range_index += 1

        if self.t_range_index > len(self.t_range) - 1:

            self.t_range_index = 0

        return self.humVal, self.tempVal
```

PART 3 – SCREENSHOTS


localhost:8888
Request one humidity-temperature reading
Request 10 humidity-temperature readings
Request stats for last 10 samples
Close down

Sensor Data Manager

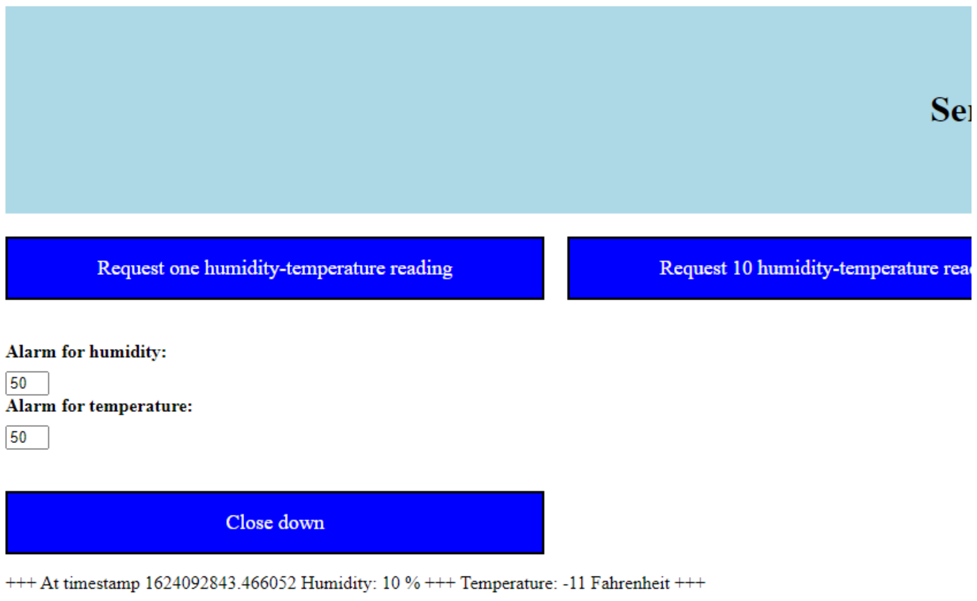
Alarm for humidity:
50
Alarm for temperature:
50



localhost:8888 says

Server or socket not available

localhost:8888
OK
Request one humidity-temperature reading
Request 10 humidity-temperature readings
Request stats for last 10 samples
Alarm for humidity:
50
Alarm for temperature:
50
Close down



Se
Request one humidity-temperature reading
Request 10 humidity-temperature rea
Alarm for humidity:
50
Alarm for temperature:
50
Close down

```
+++ At timestamp 1624092843.466052 Humidity: 10 % +++ Temperature: -11 Fahrenheit +++
```

Sensor Data Manager

Request one humidity-temperature reading | Request 10 humidity-temperature readings | Request stats for last 10 samples

Alarm for humidity:

50

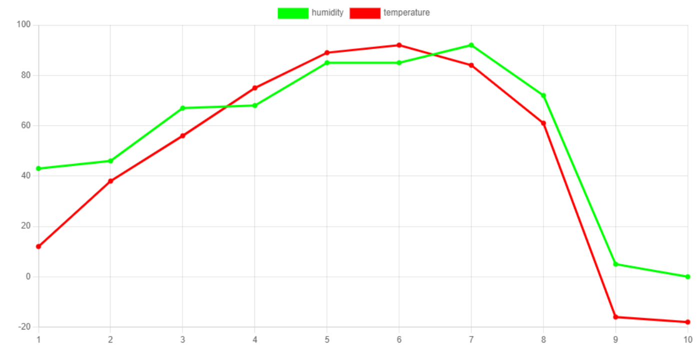
Alarm for temperature:

50

Close down

+++ Minimum Humidity 0 +++ Maximum Humidity 92 +++ Average Humidity 56.3

+++ Minimum Temperature -18 +++ Maximum Temperature 92 +++ Average Temperature 47.3



Sen

Request one humidity-temperature reading

Request 10 humidity-temperature read

Alarm for humidity:

50

Alarm for temperature:

50

Close down

+++ At timestamp 1624092843.466052 Humidity: 10 % +++ Temperature: -11 Fahrenheit +++

+++ At timestamp 1624092927.376215 Humidity: 29 % +++ Temperature: -5 Fahrenheit +++

+++ At timestamp 1624092928.394565 Humidity: 26 % +++ Temperature: 3 Fahrenheit +++

+++ At timestamp 1624092928.427014 Humidity: 41 % +++ Temperature: 17 Fahrenheit +++

+++ At timestamp 1624092928.456595 Humidity: 41 % +++ Temperature: 33 Fahrenheit +++

+++ At timestamp 1624092928.49083 Humidity: 62 % +++ Temperature: 50 Fahrenheit +++

Alarm: humidity level exceeded!

+++ At timestamp 1624092928.51381 Humidity: 67 % +++ Temperature: 71 Fahrenheit +++

Alarm: humidity level exceeded!

Alarm: temperature level exceeded!

+++ At timestamp 1624092928.531824 Humidity: 80 % +++ Temperature: 88 Fahrenheit +++

Alarm: humidity level exceeded!