SWE30011-IoT Programming
Individual Assignment

Automatic cooling fan system For closed production workshop

Name: Xinzhe Yu
Student ID: 105385294

Table of Contents

Summary

This report aims to introduce a simple IoT design to automatically control a fan system by using temperature sensor, which is usually used in a closed production workshop. In some closed automated workshops, there will be long periods of time when no staffs are involved in the work, which means the equipment will work independently. When the equipment is in operation for a long time, a lot of heat will be generated, causing the workshop temperature to rise. Ultimately, the excessively high working environment temperature will shorten the service life of the equipment. Thus, an automatic cooling fan system is extremely necessary for an automatic closed production workshop.

Equipment used in this design includes:

1. UNO R3
2. 2 buttons (digital sensor)
3. 1 temperature sensor LM35 (analog sensor)
4. 1 motor with fan
5. 3 LEDs
6. 2 resistances for safety
7. Connecting lines

This automatic fan IoT system has several functionalities:

1. It has 2 working mode which are automatic and manual. User can use a button to switch the mode of the system. The automatic mode means the fan is only controlled by the temperature sensor. When the sensor detects the temperature is higher than 30 degrees Celsius (for ease of testing, this circuit uses 26 degrees Celsius as the test threshold, but the actual operating temperature threshold is 30 degrees Celsius), the fan will be turned on automatically. For another mode which is manual mode, it means when the user clicks the button and switch into manual mode, the fan can be controlled only by the user, no matter the temperature is high or low. In addition, the mode can be switched during the fan is dunning, the state of the fan will not be affected.

2. There is button to toggle the system mode between automatic and manual.

3. There is a button which can control the fan state only during the mode is manual and this button can't control the fan when the mode is automatic.

4. There are 3 LEDs used in this design including green, red and yellow:
    a. Green LED will be on when the temperature is lower than the critical temperature.
    b. Red LED will be on when the temperature is higher than the critical temperature.
    c. When yellow LED is on, it means the system is now in manual mode. On the contrary, off means it is in automatic mode.

5. The temperature, system state and fan state data need to be saved into local database. In addition, there is a TXT file will record the time when the state of the fan changes. When an accident occurs in the workshop, such as a fire caused by overheating or equipment damage, the manager can quickly find the specific time of the accident,

thereby further confirming the cause of the accident and the responsibility. Finally, all the data in this file will be shown in the UI page.

6. The components including 2 buttons can be controlled not only physically, but also by a simple UI. The manager can use the UI to monitor the temperature changes and control the system and fan state.

Conceptual Design

This section will demonstrate the main logic of this design including design structure logic and some other technologies.

Block Diagram

This block diagram shows the structure logic of this design and how these components are combined with each other.

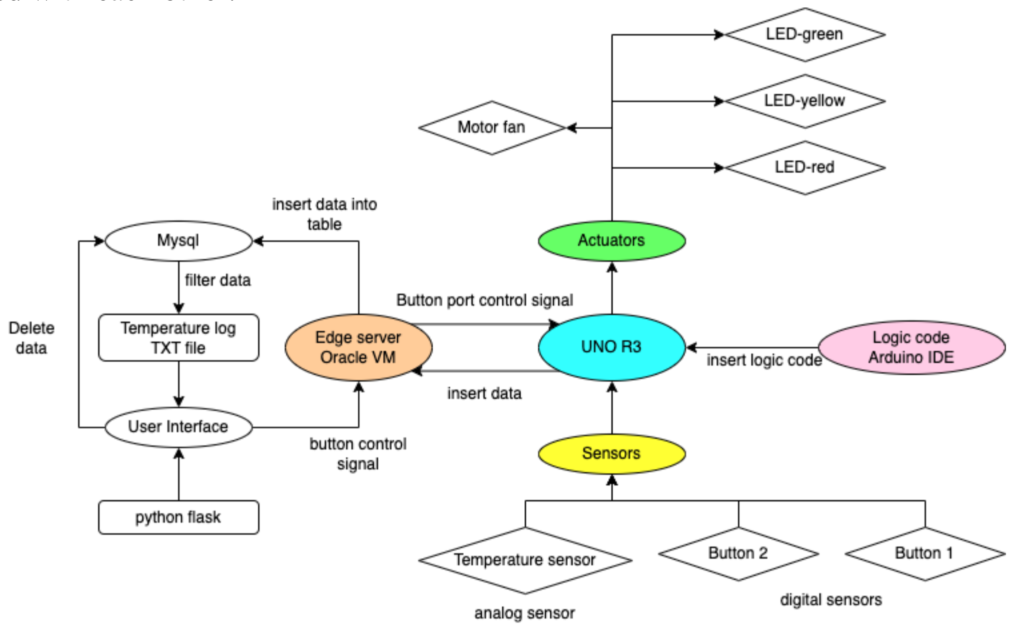


Figure 1 - Block diagram of this design

Figure 1 demonstrates the structure of this auto-fan IoT system. At first, the main logic codes are established in **Arduino IDE** and inserted into the **UNO R3** board. On the UNO R3 board, sensors including **1 temperature sensor and 2 buttons** can send the signal to control the actuators which are **3 LEDs and 1 motor fan**. At the same time, UNO R3 can send the signal data to the **edge server which is based on Oracle VM**. In this virtual machine, by using python to establish the logic, the data from UNO R3 will be saved in a table in local **Mysql**. After that, these data saved locally will be filtered and the useful data will be written into a **TXT file** as a temperature change logbook. In addition, a **user interface based on python flask** is built in edge server as well. In the page of this simple UI, user can not only check the data saved in temperature logbook and check the sensors' states, but also send the signal to the UNO R3 to control 2 buttons' state.

UML

This is the UML to show how different components used in this IoT system works together:

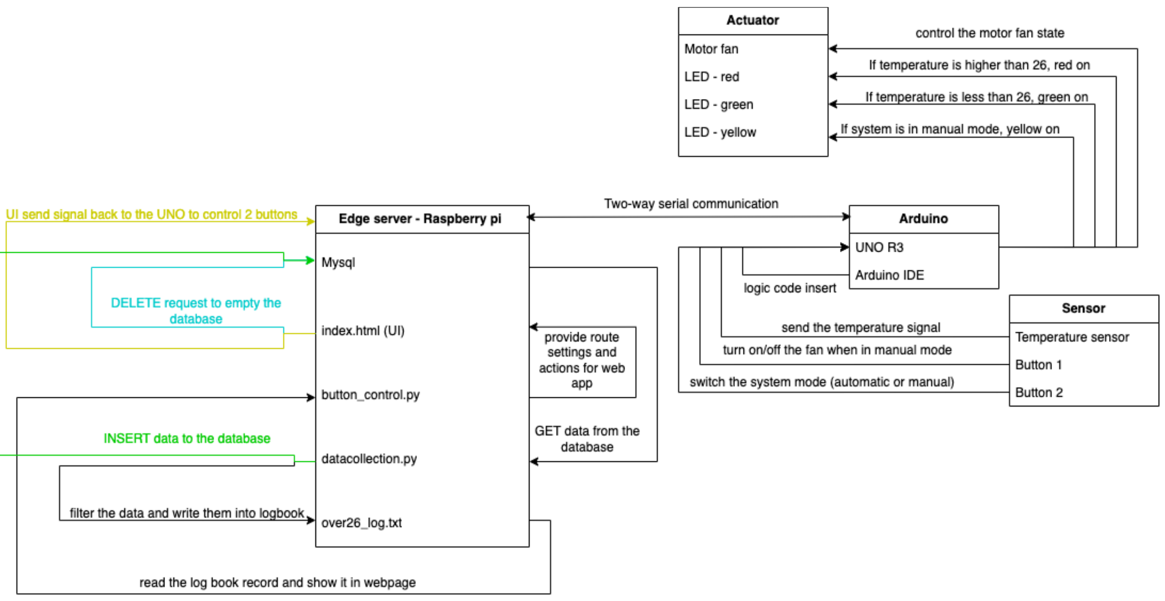

Actuator
Motor fan
LED - red
LED - green
LED - yellow
control the motor fan state
If temperature is higher than 26, red on
If temperature is less than 26, green on
If system is in manual mode, yellow on
Two-way serial communication
Arduino
UNO R3
Arduino IDE
logic code insert
Sensor
Temperature sensor
Button 1
Button 2
send the temperature signal
turn on/off the fan when in manual mode
switch the system mode (automatic or manual)
UI send signal back to the UNO to control 2 buttons
Edge server - Raspberry pi
Mysql
index.html (UI)
button_control.py
datacollection.py
over26_log.txt
DELETE request to empty the database
INSERT data to the database
filter the data and write them into logbook
provide route settings and actions for web app
GET data from the database
read the log book record and show it in webpage

Figure 2 – UML of this design

Figure 2 shows more details about the system, especially in edge server. There are 4 files I built for this system:

1. **index.html**: uses html code to establish a simple webpage to show the UI.
2. **button_contrl.py**: serial connects to the UNO, uses the framework Flask to build the local server and provide the action for each route.
3. **datacollection.py**: serial connects to the UNO, insert the data from UNO into the database and record the time and the temperature when the fan state changes.
4. **over26_log.txt**: the logbook to record the time and the temperature when the fan state changes.

Implementation

This part will explain the how sensors and actuators work in this IoT system. In addition, the library and software will be introduced as well.

Sensors

This project use 1 analog temperature sensor (LM35) and 2 digital button sensors.

a. LM35:
   The LM35is used to monitor the environment temperature and send the signal directly to the UNO R3. The reason why to monitor the temperature is when the temperature is higher than the critical temperature, the working environment is too hot for equipment working. At this time, the fan needs to be turned on automatically to cool down the workshop working environment temperature. In the code, the LM35 is connected with A0.

b. Button 1:
Button 1 is used to monitor the push behaviour and change the system mode (automatic and manual). Automatic mode means only LM35 can control the fan and manual mode means only users can control the fan. In the code, this button is connected with D7 and will provide a digital signal to control the LED-yellow and the mode switch.

c. Button 2:
Button 2 is used to monitor the push behaviour and change the state of the fan only if the system mode is in manual (Button 1 is set). In the code, button 2 is connected with D6 and will provide a digital signal to control the on and off of the motor fan.

Actuators

There are 3 LEDs and 1 motor fan are used as the actuators in this system:

a. LED-green: to represent the temperature. If the working temperature is lower than the critical temperature, the green LED is on. In the code, the green LED is connected with D3 pin.

b. LED-red: to represent the temperature. If the working temperature is higher than the critical temperature, the red LED is on. In the code, the red LED is connected with D2 pin.

c. LED-yellow: to represent the system mode (automatic or manual). If the system working mode is automatic, the yellow LED is off and if it is manual, the yellow LED is on. In the code, the yellow LED is connected with D4 pin.

I connect LEDs with the pin on UNO and use the code to control the LOW or HIGH of the pin, which can set the LED off or on.

d. Motor fan: to cool down the temperature of the working environment. It can be controlled by users manually or LM35 sensor automatically, just depends on the mode. In the code, the motor is connected with D5.

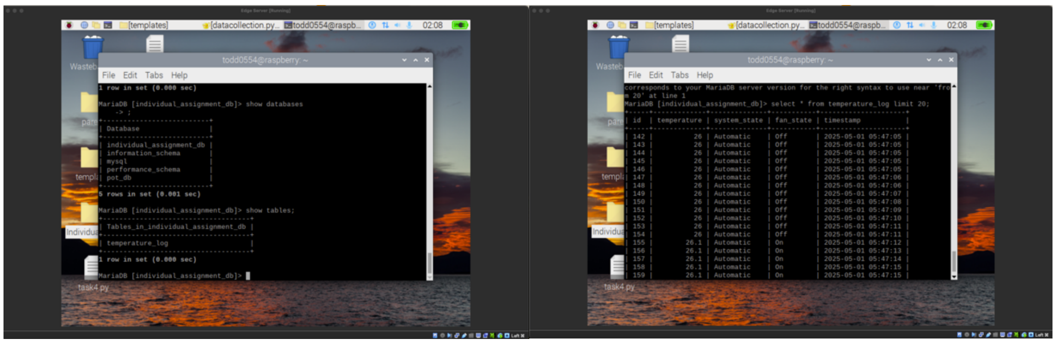
Software/Libraries

There are some other softwares used in this design, which are Arduino IDE, VirtualBox, MYSQL, Python flask framework and several libraries in python.

1. Arduino IDE: a platform to code and insert the logic code into UNO R3.

2. VirtualBox: it is used to provide a platform for raspberry pi server and this server hold the database and the webserver of the UI.

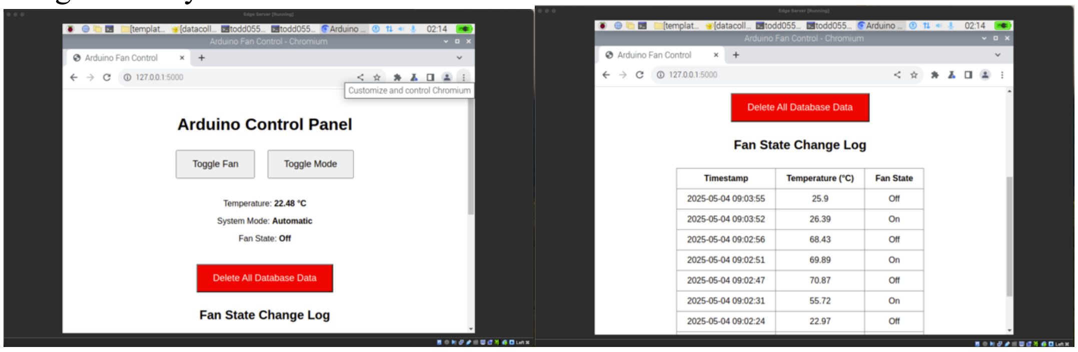
3. MYSQL: it is built in raspberry pi server and provide a table structure to store all the data sent from the UNO R3. The data from the UNO will be stored into a databases called individual_assignment_db and the table is called temperature_log. Here are details of the database:


```
1 row in set (0.000 sec)

MariaDB [individual_assignment_db]> show databases
    -> ;
+--------------------------+
| Database                 |
+--------------------------+
| individual_assignment_db |
| information_schema       |
| mysql                    |
| performance_schema       |
| pot_db                   |
+--------------------------+
5 rows in set (0.001 sec)

MariaDB [individual_assignment_db]> show tables;
+------------------------------------+
| Tables_in_individual_assignment_db |
+------------------------------------+
| temperature_log                    |
+------------------------------------+
1 row in set (0.000 sec)

MariaDB [individual_assignment_db]> 
```

```
corresponds to your MariaDB server version for the right syntax to use near 'fro
m 20' at line 1
MariaDB [individual_assignment_db]> select * from temperature_log limit 20;
+-----+-------------+--------------+-----------+---------------------+
| id  | temperature | system_state | fan_state | timestamp           |
+-----+-------------+--------------+-----------+---------------------+
| 142 |          26 | Automatic    | Off       | 2025-05-01 05:47:05 |
| 143 |          26 | Automatic    | Off       | 2025-05-01 05:47:05 |
| 144 |          26 | Automatic    | Off       | 2025-05-01 05:47:05 |
| 145 |          26 | Automatic    | Off       | 2025-05-01 05:47:05 |
| 146 |          26 | Automatic    | Off       | 2025-05-01 05:47:05 |
| 147 |          26 | Automatic    | Off       | 2025-05-01 05:47:06 |
| 148 |          26 | Automatic    | Off       | 2025-05-01 05:47:06 |
| 149 |          26 | Automatic    | Off       | 2025-05-01 05:47:07 |
| 150 |          26 | Automatic    | Off       | 2025-05-01 05:47:08 |
| 151 |          26 | Automatic    | Off       | 2025-05-01 05:47:09 |
| 152 |          26 | Automatic    | Off       | 2025-05-01 05:47:10 |
| 153 |          26 | Automatic    | Off       | 2025-05-01 05:47:11 |
| 154 |          26 | Automatic    | Off       | 2025-05-01 05:47:11 |
| 155 |        26.1 | Automatic    | On        | 2025-05-01 05:47:12 |
| 156 |        26.1 | Automatic    | On        | 2025-05-01 05:47:13 |
| 157 |        26.1 | Automatic    | On        | 2025-05-01 05:47:14 |
| 158 |        26.1 | Automatic    | On        | 2025-05-01 05:47:15 |
| 159 |        26.1 | Automatic    | On        | 2025-05-01 05:47:15 |
```

Figure 3 – MYSQL details in Raspberry PI server

4. Python flask: a framework to build the local webserver and the frontend UI for this design. The layout is shown below:



Arduino Control Panel

Temperature: **22.48 °C**
System Mode: **Automatic**
Fan State: **Off**

Fan State Change Log

Timestamp	Temperature (°C)	Fan State
2025-05-04 09:03:55	25.9	Off
2025-05-04 09:03:52	26.39	On
2025-05-04 09:02:56	68.43	Off
2025-05-04 09:02:51	69.89	On
2025-05-04 09:02:47	70.87	Off
2025-05-04 09:02:31	55.72	On
2025-05-04 09:02:24	22.97	Off

Figure 4 – Layout of the UI

5. serial library: it is used to select the hardware port of the computer and read the signal from UNO.

6. pymysql: this library is used to connect the MYSQL and insert SQL code into the MYSQL database from Python environment.

7. time: it is used to set a time sleep after some codes which need more time to run.

8. datetime: this library is used to get the real time now.

Resources

I used the knowledges from our tutorials including Tutorial 2, 3, 4, 6 and 7.

Appendix

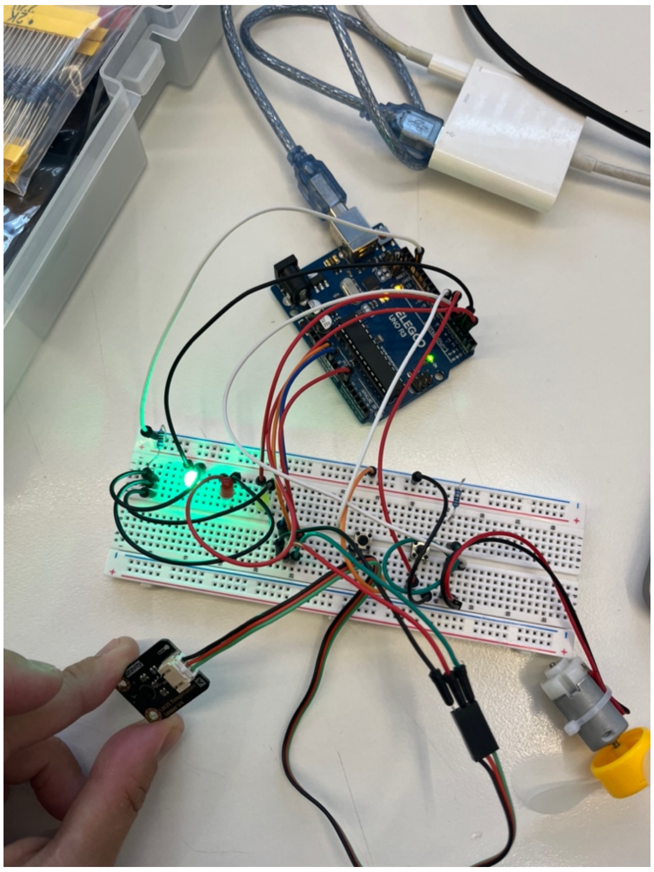
Presentation video link

https://www.youtube.com/watch?v=cm7A-DU45UU

Github

https://github.com/Todd0554/SWE30011-IoT-Programming-Individual-Assignment

The connection of the breadboard



Arduino IDE code

```
#define tempSensorPin A0
#define fanButtonPin 6
#define stateButtonPin 7

const int redLEDPin = 2;
const int greenLEDPin = 3;
const int yellowLEDPin = 4;
const int fanPin = 5;

bool manualFanOverride = true;
bool lastFanButtonState = HIGH;
bool manualStateOverride = false;
bool lastStateButtonState = HIGH;

void setup() {
  pinMode(redLEDPin, OUTPUT);
  pinMode(greenLEDPin, OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
  pinMode(fanPin, OUTPUT);
  pinMode(fanButtonPin, INPUT_PULLUP);
  pinMode(stateButtonPin, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  int analogValue = analogRead(tempSensorPin);
  float voltage = analogValue * (5.0 / 1023.0);       // Convert to voltage
  float temperature = voltage * 100.0;                // LM35: 10mV = 1°C

  if (temperature < -10.0 || temperature > 80.0) {
    return;
  }

  bool currentStateButtonState = digitalRead(stateButtonPin);
  if (lastStateButtonState == HIGH && currentStateButtonState == LOW) {
    manualStateOverride = !manualStateOverride;
  }
  lastStateButtonState = currentStateButtonState;

  Serial.print("DATA:");
  Serial.print(temperature);
  Serial.print(",");
  Serial.print(manualStateOverride ? "Manual" : "Automatic");
  Serial.print(",");

  digitalWrite(yellowLEDPin, LOW);
  if (!manualStateOverride){
    if (temperature > 28.0) {
      digitalWrite(fanPin, HIGH);
      digitalWrite(greenLEDPin, LOW);
      digitalWrite(redLEDPin, HIGH);
      if(manualFanOverride == false){
        manualFanOverride = true;
      }
    } else {
      digitalWrite(fanPin, LOW);
      digitalWrite(greenLEDPin, HIGH);
      digitalWrite(redLEDPin, LOW);
      if(manualFanOverride == true){
        manualFanOverride = false;
      }
    }
  } else {
    bool currentFanButtonState = digitalRead(fanButtonPin);
    if (lastFanButtonState == HIGH && currentFanButtonState == LOW) {
      manualFanOverride = !manualFanOverride;
    }
    lastFanButtonState = currentFanButtonState;
    if (manualFanOverride){
      digitalWrite(fanPin, HIGH);
      digitalWrite(yellowLEDPin, HIGH);
    } else {
      digitalWrite(fanPin, LOW);
      digitalWrite(yellowLEDPin, HIGH);
    }
  }

  Serial.println(manualFanOverride ? "On" : "Off");

  if (Serial.available()) {
    String command = Serial.readStringUntil('\n');
    command.trim();

    if (command == "FAN_TOGGLE") {
      manualFanOverride = !manualFanOverride;
    } else if (command == "MODE_TOGGLE") {
      manualStateOverride = !manualStateOverride;
    }
  }
  delay(800);
}
```

Codes for MYSQL connecting and data inserting (datacollection.py)

```
import serial
import pymysql
import time
from datetime import datetime

# setting up serial connection
ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(3)  # give Arduino time to prepare

# database connection
db = pymysql.connect(
    host="localhost",
    user="xinzhe",
    password="",
    database="individual_assignment_db"
)
cursor = db.cursor()
```

```
# create table if not exists
cursor.execute("""
CREATE TABLE IF NOT EXISTS temperature_log (
    id INT AUTO_INCREMENT PRIMARY KEY,
    temperature FLOAT,
    system_state VARCHAR(10),
    fan_state VARCHAR(10),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
""")

# initialize log file
log_file = "over26_log.txt"

try:
    while True:
        line = ser.readline().decode(errors='ignore').strip()
        if not line:
            continue

        if line.startswith("DATA:"):
            try:
                data = line[5:].split(",")
                temperature = float(data[0])
                mode = data[1]
                fan_state = data[2]

                # insert into table
                cursor.execute("INSERT INTO temperature_log (temperature,
system_state, fan_state) VALUES (%s, %s, %s)",
                               (temperature, mode, fan_state))
                db.commit()
                print(f"✓ Saved to DB: {temperature:.1f}°C | {mode} | Fan
{fan_state}")

                # write into log file if temperature > 26.0
                if temperature > 26.0:
                    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    with open(log_file, "a") as file:
                        file.write(f"{timestamp} - Temp: {temperature:.1f}°C\n")
                    print(f"write into TXT(>26°C):{temperature:.1f}°C")

            except Exception as e:
                print("Data extracting error:", line, "→", e)

except KeyboardInterrupt:
    print("\nprepare to exit...")

finally:
    ser.close()
    cursor.close()
    db.close()
    print("Exiting successfully")
```

Codes of UI logic in button_control.py

```
from flask import Flask, render_template, redirect, url_for, request
import pymysql
import serial
import time
from datetime import datetime

app = Flask(__name__)

ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
time.sleep(5)

DB_CONFIG = {
    'host': 'localhost',
    'user': 'xinzhe',
    'password': '',
    'database': 'individual_assignment_db'
}

def get_arduino_status():
    ser.reset_input_buffer()
    time.sleep(0.5)
    start_time = time.time()
    while time.time() - start_time < 1.0:
        line = ser.readline().decode(errors='ignore').strip()
        if line.startswith("DATA:"):
            try:
                parts = line[5:].split(",")
                temperature = float(parts[0])
                mode = parts[1]
                fan = parts[2]
                return {"temperature": temperature, "mode": mode, "fan": fan}
            except:
                continue
    return {"temperature": "--", "mode": "--", "fan": "--"}

def get_fan_changes():
    conn = pymysql.connect(**DB_CONFIG)
    cursor = conn.cursor()
    cursor.execute("""
        SELECT timestamp, temperature, fan_state
        FROM temperature_log
        WHERE id IN (
            SELECT id FROM (
                SELECT id, fan_state,
                    LAG(fan_state) OVER (ORDER BY timestamp) AS prev_state
                FROM temperature_log
            ) AS t
            WHERE fan_state != prev_state OR prev_state IS NULL
        )
        ORDER BY timestamp DESC
        LIMIT 10
    """)
    rows = cursor.fetchall()
    conn.close()
    return rows

@app.route('/')
def index():
    status = get_arduino_status()
    fan_data = get_fan_changes()
    return render_template('index.html', fan_data=fan_data, **status)

@app.route('/delete_all', methods=['POST'])
def delete_all():
    conn = pymysql.connect(**DB_CONFIG)
    cursor = conn.cursor()
    cursor.execute("DELETE FROM temperature_log")
    conn.commit()
    conn.close()
    return redirect(url_for('index'))

@app.route('/toggle_fan')
def toggle_fan():
    ser.write(b'FAN_TOGGLE\n')
    time.sleep(3)
    return redirect(url_for('index'))

@app.route('/toggle_mode')
def toggle_mode():
    ser.write(b'MODE_TOGGLE\n')
    time.sleep(3)
    return redirect(url_for('index'))

if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000, debug=True)
```

Codes of the layout of the UI in html in index.html (located in templates folder)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Arduino Fan Control</title>
    <style>
```

```html
        body { font-family: Arial; text-align: center; margin-top: 50px; }
        button { padding: 15px 30px; font-size: 18px; margin: 10px; }
        table { margin: auto; border-collapse: collapse; margin-top: 30px; }
        th, td { border: 1px solid #888; padding: 10px 20px; }
    </style>
</head>
<body>
    <h1>Arduino Control Panel</h1>

    <form action="/toggle_fan" method="get" style="display:inline;">
        <button type="submit">Toggle Fan</button>
    </form>
    <form action="/toggle_mode" method="get" style="display:inline;">
        <button type="submit">Toggle Mode</button>
    </form>

    <div class="status" style="margin-top: 30px;">
        <p>Temperature: <strong>{{ temperature }} °C</strong></p>
        <p>System Mode: <strong>{{ mode }}</strong></p>
        <p>Fan State: <strong>{{ fan }}</strong></p>
    </div>

    <form action="/delete_all" method="post" style="margin-top: 30px;">
        <button type="submit" style="background-color:red; color:white;">Delete All
Database Data</button>
    </form>

    <h2>Fan State Change Log</h2>
    <table>
        <tr>
            <th>Timestamp</th>
            <th>Temperature (°C)</th>
            <th>Fan State</th>
        </tr>
        {% for row in fan_data %}
        <tr>
            <td>{{ row[0] }}</td>
            <td>{{ row[1] }}</td>
            <td>{{ row[2] }}</td>
        </tr>
        {% endfor %}
    </table>


</body>
</html>
```