

INDIVIDUAL ASSIGNMENT (PRACTICAL)

How IOT contribute to agriculture development

Author: Tran Thanh Minh

Student ID: 103809048

COS30011 – IoT Programming

Fall 2023

Lecturer: Tuan Tran

Swinburne University of Technology

Contents

I.	Summary.....	3
a.	Topic background	3
b.	Proposed system	4
II.	Conceptual Design	6
a.	Block diagrams	6
b.	UML diagram.....	7
III.	Implementation.....	10
a.	Sensors	10
b.	Actuators.....	10
c.	Software/Libraries:	10
IV.	Evidence of system.....	12
a.	Tinkercad Model.....	12
b.	Evidence of building system	13
c.	Evidence of full process.....	13
V.	Resources.....	28
VI.	Appendix	29

I. Summary

a. Topic background

The emergence of the Internet of Things (IoT) has ushered in a transformative era, reshaping industries across the globe. One sector profoundly impacted is agriculture, where the interconnectivity of devices and data-driven practices are revolutionizing traditional farming methods. In this context, my survey paper delves into the multifaceted influence of IoT innovation on agriculture, exploring its implications from precision farming to sustainable practices.

Traditionally, agriculture has been characterized by tradition and manual labor. However, with the surge in global population and escalating environmental concerns, the necessity for more efficient, sustainable, and data-driven approaches to farming has become increasingly apparent. The integration of IoT into agriculture provides a technological frontier that addresses age-old challenges. This paper investigates how IoT enables farmers to remotely oversee their land and crops, receiving critical updates and insights on their smartphones or personal devices. This level of connectivity affords an unprecedented level of control and responsiveness, empowering farmers and citizens alike to take timely actions to mitigate potential risks and losses.

In the realm of farming practices, the integration of IoT technologies has ushered in a new era of agricultural management and efficiency. Sustainable farming practices have been significantly bolstered by IoT. The continuous monitoring of environmental conditions facilitated by IoT assists in pest management and disease prevention.

A proposed smart agriculture system for cornfields, utilizing wireless sensor networks and drones, is scrutinized for its benefits and limitations. The need for many sensors and the cost of drones are identified as challenges, despite the potential to improve crop yields.

Lastly, the survey explores a smart farming system based on IoT sensors for data collection and cloud computing for analysis. Machine learning is employed to provide farmers with recommendations on enhancing crop yields and reducing environmental impact. Emphasizing the real-time data collection capabilities of IoT sensors, enabling farmers to make informed decisions for improved crop yields, reduced costs, and environmental sustainability. While acknowledging

the transformative power of IoT, the survey paper also highlights existing challenges such as high costs and complexity that need addressing for widespread adoption in agriculture. As IoT continues to evolve, its role in agriculture is poised to grow, contributing to a more efficient, profitable, and sustainable future for farmers and industry.

b. Proposed system

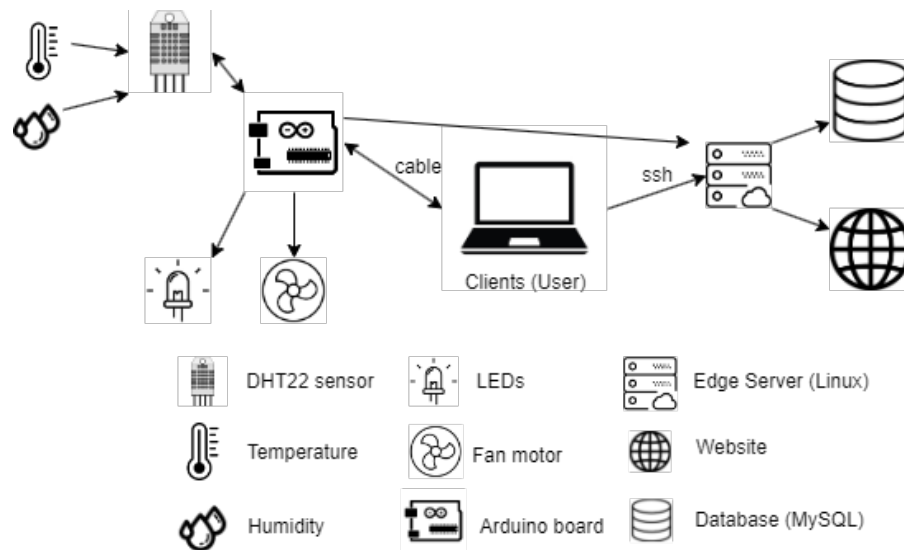


Figure 1: Sketch of proposed system

The proposed system consists of the following components:

- Sensors: The sensors collect data about the environment, such as temperature and humidity.
- Edge server: the edge server collects data from the sensors and processes it in real time. It also can be used to write serial input into the Arduino board.
- Database: The database stores the data collected by the sensors and the edge server.

The clients can interact with the system through web browsers. This proposed system offers several benefits including Real-time monitoring, distributed architecture, and scalability. The system can monitor the real-time and detect changes immediately, allowing the system to respond to changes quickly and alert through actuators. The components of the system are in different places and make the system more scalable and reliable and can be expanded by adding more sensors and actuators. This system can help to monitor and control devices in an agriculture

setting such as irrigation systems and greenhouses. This system also involves the use of storage MySQL and can store both humidity, temperature, and current time stamp for managing.

For the sensor for collecting humidity and temperature, DHT22 is suitable for this use case, but humidity will be mainly used for controlling the system. Humidity is about 60% - 80% is ideal for plants (PLNTS.com, 2021) but in this assignment I will change the humidity for trigger conditions so that changes in humidity can be easier to managed.

II. Conceptual Design

a. Block diagrams

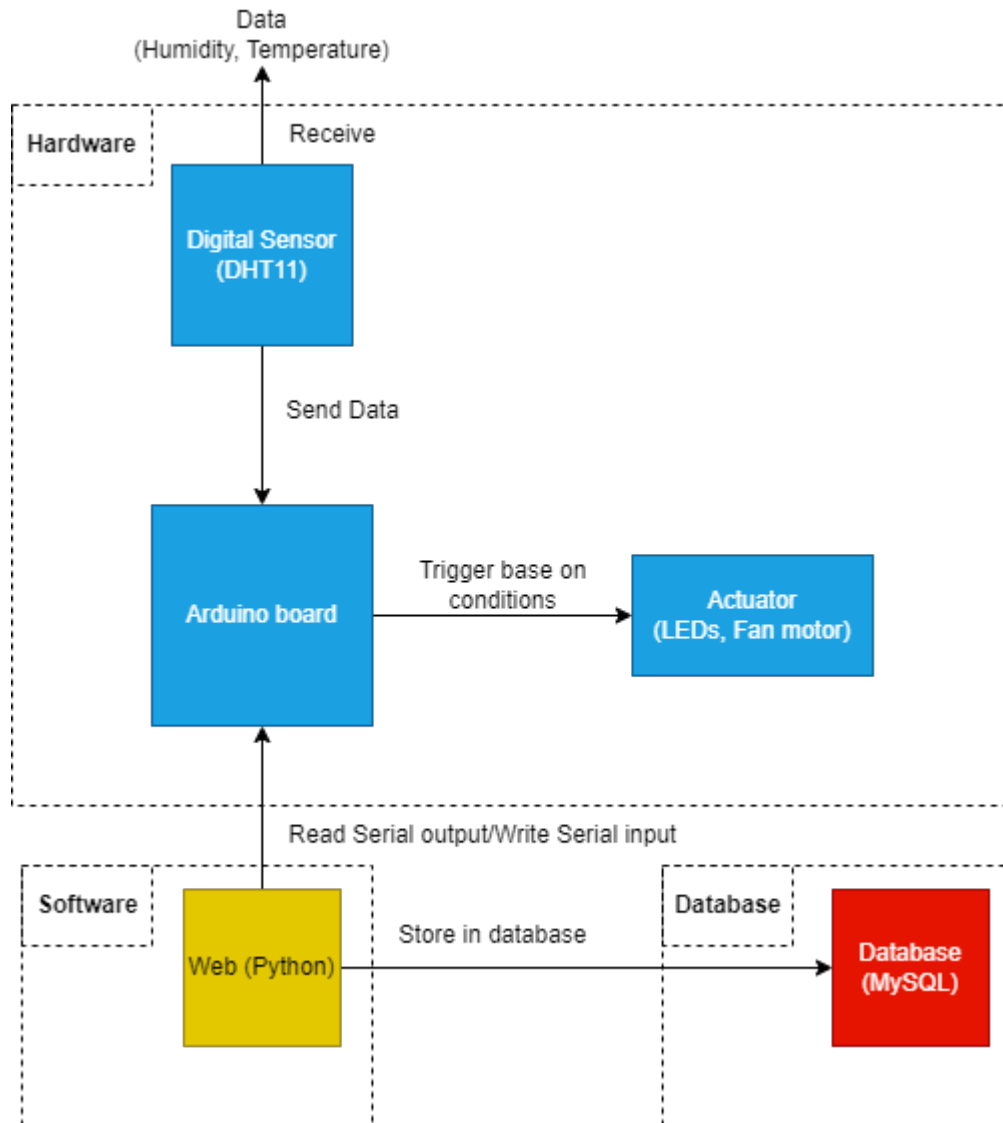


Figure 2: Block diagram for proposed system

First, the digital sensor (DHT11) will receive the data (Humidity and Temperature) from the environment then it will send the data to the Arduino board. Based on the data and conditions, the Arduino board will trigger the actuator (LEDs) for notifying. The web application can show the

front end and read the serial output from the Arduino board and display the data. Depending on the user, the web application can perform the action to store the data to the database (MySQL).

b. UML diagram

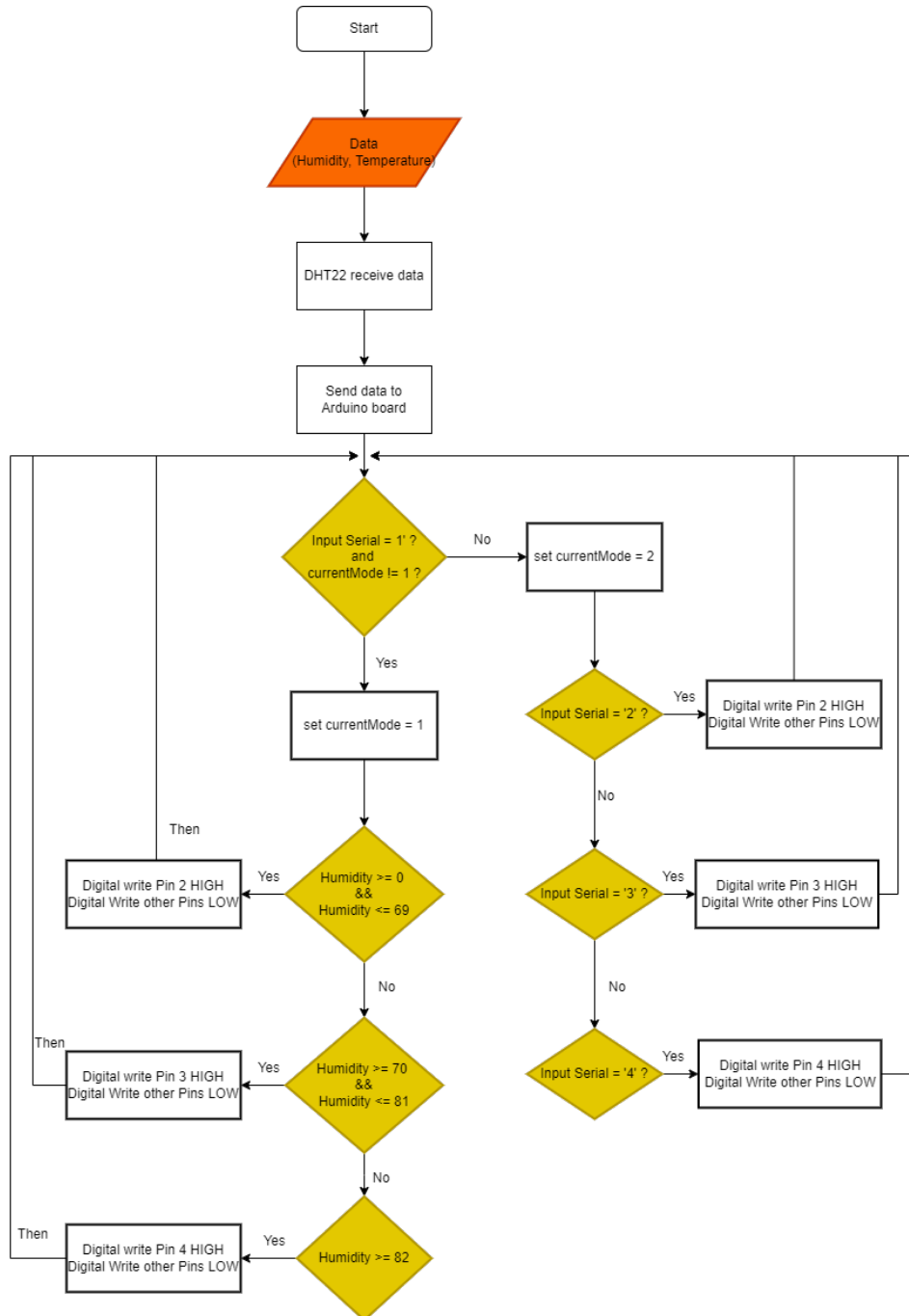


Figure 3: Flow chart for how the Arduino work

For the above chart, the Arduino code runs in the order and keeps looping. The data are humidity and temperature will be received from the DHT22. It can be divided into 2 modes: for the first mode when the user input '1' to the Arduino, the code will automatically run and detect the humidity it receives from the DHT22 sensor. If the user enters '2' or '3' or '4' to the Arduino, the current mode will switch to 2 and perform the action of setting the corresponding pin to HIGH and other pins to LOW state.

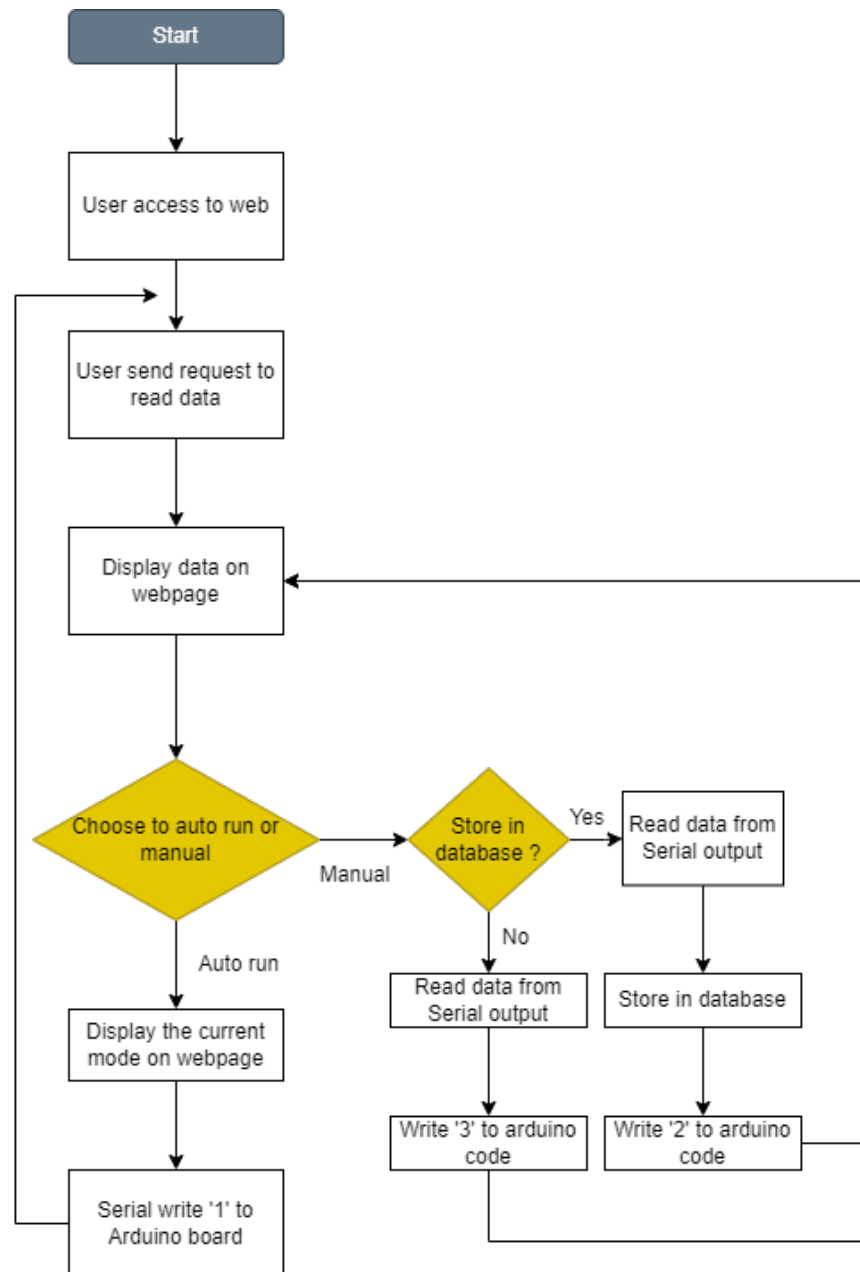


Figure 4: Flow chart for how the python program works and send request.

For the above image, the description of the system is visualized. First, when the user accesses the web, the webpage will display and then when the user requests the data, the data will be fetched by Python and display back in the web. The user can choose whenever the Arduino runs automatically or manually for managing the state. If the user chooses to run automatically, the back end of the website will write the Serial input to the Arduino board and the program will auto run (refer to Figure 3). However, if the user chooses to run manually, the user can choose whenever to store the data in the database or not. In both cases, the first thing the backend of the web does is to read the data (humidity and temperature) from the Serial output. If the user chooses to store, the backend of the web will store the data (humidity, temperature) in the database include the time and date of the current request. the backend of the web will send Serial input '2' to the Arduino code and then the Arduino board will trigger the suitable pin to alert (refer to Figure 4).

III. Implementation

The Arduino servers as a node, collecting and processing data from sensors and sending it to the edge server. The edge server processes this data further and communicates with a high-level system and displays it on web page. The edge server powered by Python and Flask communicates with Arduino through serial communication, fetching sensor data and sending control signals.

a. Sensors

DHT is a sensor for measuring the temperature and humidity of the surrounding environment. In this project, DHT is used to monitor the environmental conditions. This is crucial for applications such as climate control or data logging where knowing temperature and humidity is essential. The DHT22 is integrated with Arduino, which reads the sensor values and prints out Serial output in the customized format. It can take actions based on the data retrieved.

b. Actuators

1. LED is used in this project for status display. It can help to signal the specific conditions when the data received meets one of the conditions such as turning on it when a certain temperature is reached. It is integrated with the Arduino and controlled by the program logic to provide visual feedback.
2. Fan motor serves as drying the humidity based on the humidity readings from the DHT22. If the humidity goes over the at some point, the fan motor activates to dry the environment. Integrating with the Arduino enables automated humidity control and is crucial for real-life applications like cooling systems.

c. Software/Libraries:

Python is one of the programming languages used in this project along with its libraries like MySQLdb, serial, Flask, render_template is used for employing data processing, communication and front-end. Python interacts with the Arduino via the serial port, retrieving sensor and storing it in a MySQL database. Flask and render_template will facilitate the creation of a web interface for users to monitor and control the system by sending requests to the programmed endpoints.

MySQL will serve as the database to store sensor data. This relational database system allows efficient management and retrieval of information which will support storing and managing all the records including time and date information. This can help for analyzing the temperature and humidity patterns.

IV. Evidence of system

a. Tinkercad Model

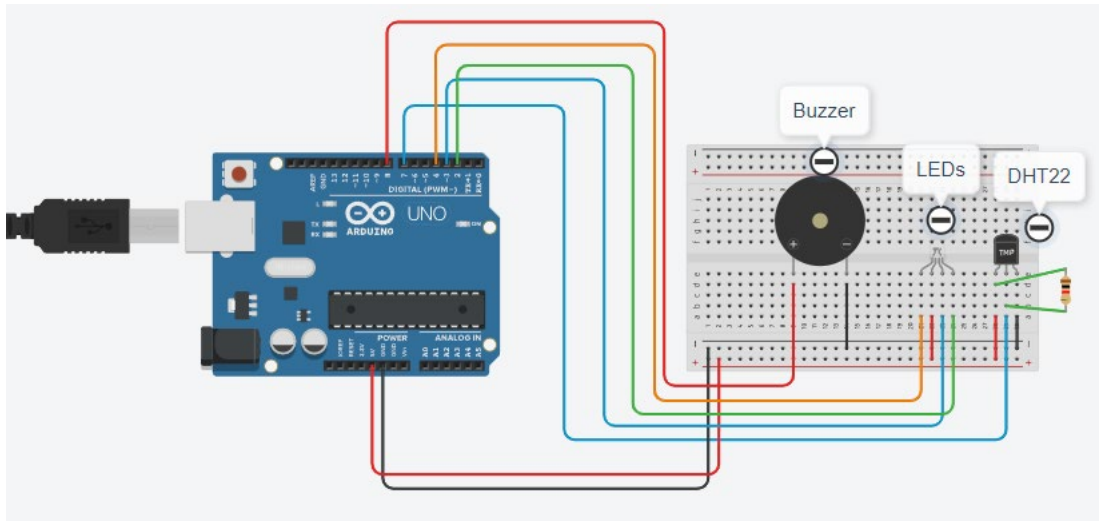


Figure 5: Tinkercad Model of proposed system

Due to Tinkercad doesn't have the suitable motor for my proposed system, so I have replaced it with the buzzer for displaying. However, the functionality is still the same. The system will alert based on the conditions of the code. Below is the schematic view which includes the logic of the circuit.

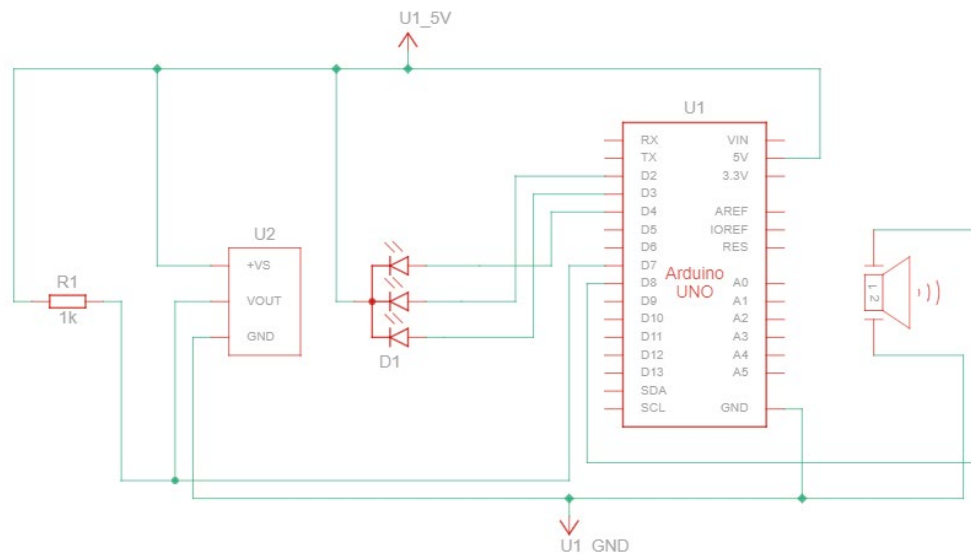


Figure 6: Schematic view of Tinkercad

b. Evidence of building system

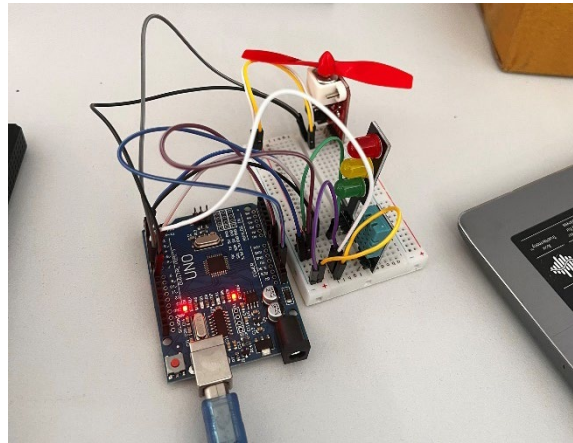


Figure 7: Side view of proposed system

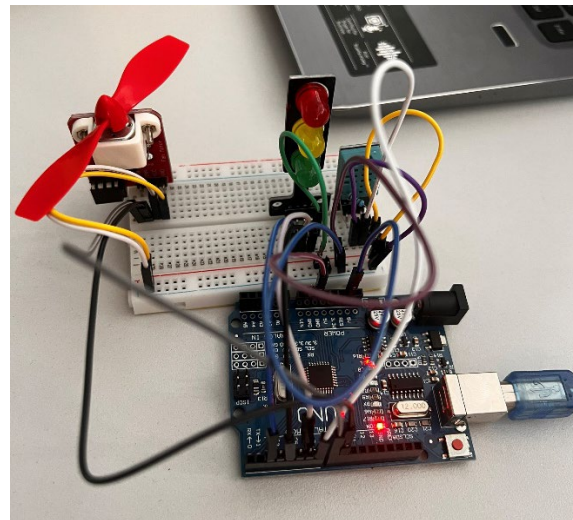


Figure 8: Front view of proposed system

c. Evidence of full process

Base URL: <http://192.168.153.131/>

Command:

Note: If there is any process fail, the Arduino will receive the Serial input 4 and will trigger red light for alerting the error.

- Get current data. (URL/getData)
 - o This will get the current data from the sensor and display it on the web.

Arduino Web Server

Get the Data

Get the current data

Humidity: 73.0

Temperature: 25.0

Figure 9: Get current data command.

- Run automatically. (URL/automatically)
 - o This will write the Serial input 1 into the Arduino code and the system will run automatically based on the logic.

Run Automatically

Note: Database won't be stored if it is automatically run

Run Automatically

Figure 10: Automatically run command.

- Get data and store data into database. (URL/storeData)
 - o This will include getting current data and storing it into the database. It also will write the Serial input 2 into the Arduino board and trigger green light.

Database

is currently **on**

Store Data

No Store Data

Figure 11: Store database is currently on

- Get data and no store data into database. (URL/noStoreData)
 - o Get the current data and turn the mode of database to off.

Database

is currently **off**

Store Data

No Store Data

Figure 11: Current database is off.

- Get data from database. (URL/getDatabase)
 - o Get the records from the database and display them on the web page.

Data from Database

Note: Time may not the same with your local time due to server configuration

Maximum display the last 10 record of data

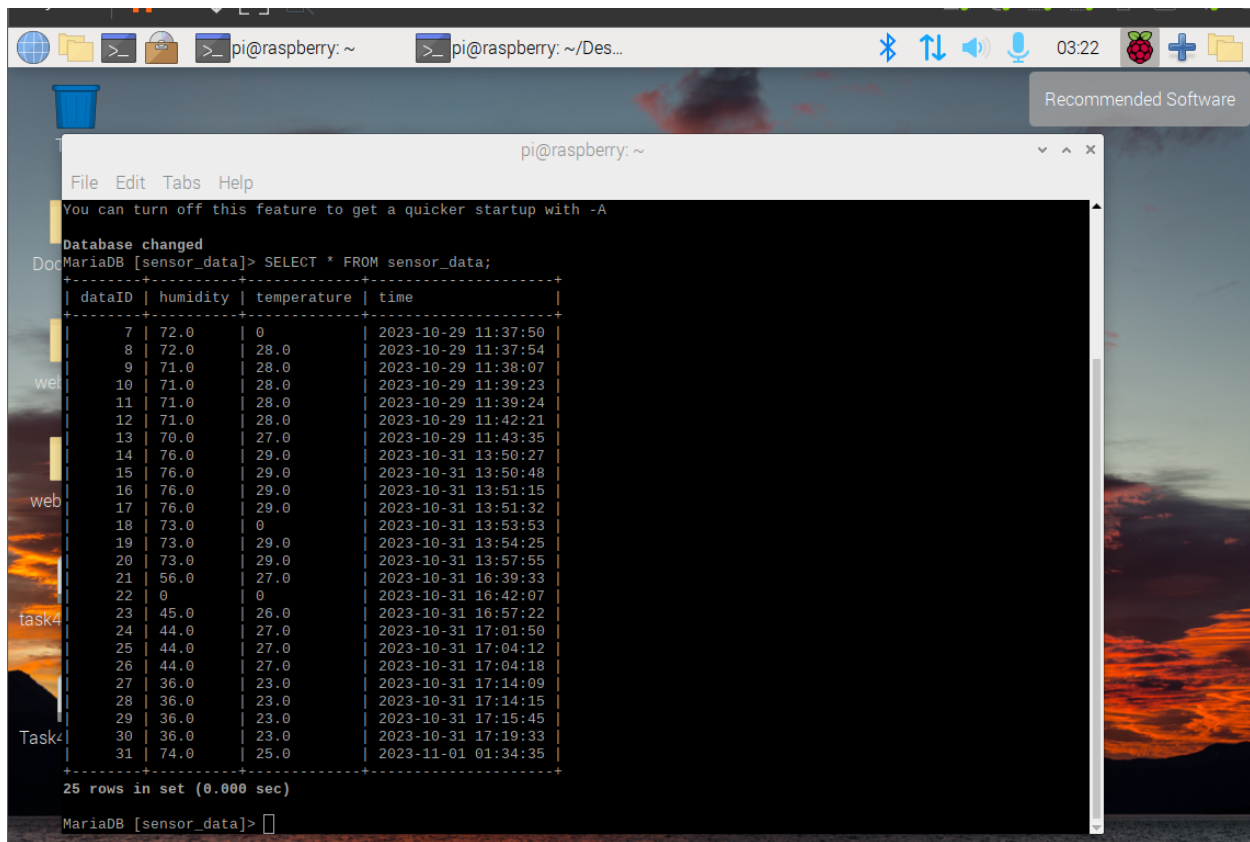
You need to store the data in the database first so it can get data from database

Get data from database

Humidity	Temperature	Date and Time
74.0	25.0	2023-11-01 01:34:35
36.0	23.0	2023-10-31 17:19:33
36.0	23.0	2023-10-31 17:15:45
36.0	23.0	2023-10-31 17:14:15
36.0	23.0	2023-10-31 17:14:09
44.0	27.0	2023-10-31 17:04:18
44.0	27.0	2023-10-31 17:04:12
44.0	27.0	2023-10-31 17:01:50
45.0	26.0	2023-10-31 16:57:22
0	0	2023-10-31 16:42:07

Figure 12: Table for displaying records.

Database of records:

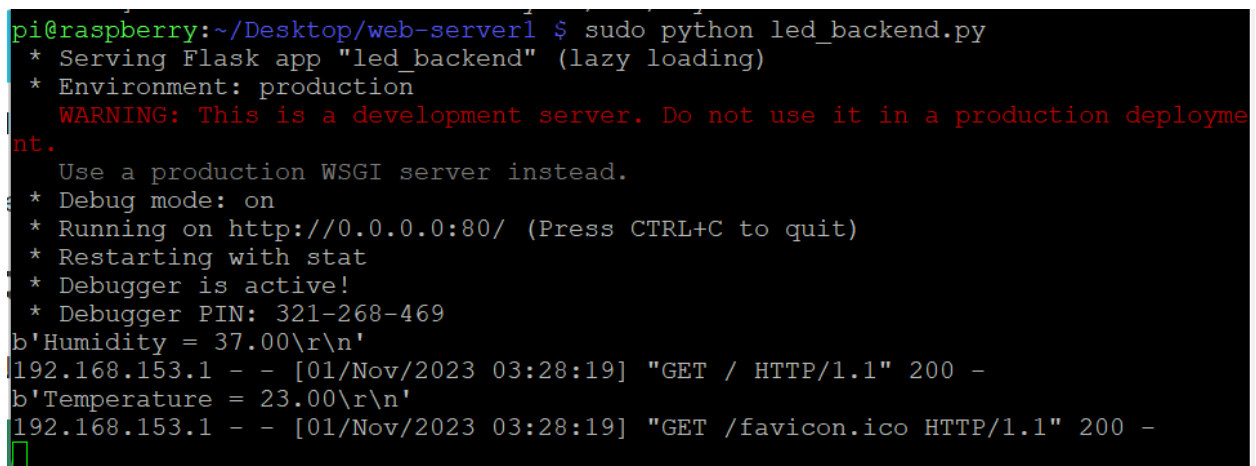


The screenshot shows a Raspberry Pi desktop environment. A terminal window is open, displaying a MariaDB query result. The query is `SELECT * FROM sensor_data;` and the result is a table with 25 rows. The table has four columns: `dataID`, `humidity`, `temperature`, and `time`. The data shows sensor readings from October 29 to November 1, 2023. The terminal window also shows a message: "Database changed" and "25 rows in set (0.000 sec)".

dataID	humidity	temperature	time
7	72.0	0	2023-10-29 11:37:50
8	72.0	28.0	2023-10-29 11:37:54
9	71.0	28.0	2023-10-29 11:38:07
10	71.0	28.0	2023-10-29 11:39:23
11	71.0	28.0	2023-10-29 11:39:24
12	71.0	28.0	2023-10-29 11:42:21
13	70.0	27.0	2023-10-29 11:43:35
14	76.0	29.0	2023-10-31 13:50:27
15	76.0	29.0	2023-10-31 13:50:48
16	76.0	29.0	2023-10-31 13:51:15
17	76.0	29.0	2023-10-31 13:51:32
18	73.0	0	2023-10-31 13:53:53
19	73.0	29.0	2023-10-31 13:54:25
20	73.0	29.0	2023-10-31 13:57:55
21	56.0	27.0	2023-10-31 16:39:33
22	0	0	2023-10-31 16:42:07
23	45.0	26.0	2023-10-31 16:57:22
24	44.0	27.0	2023-10-31 17:01:50
25	44.0	27.0	2023-10-31 17:04:12
26	44.0	27.0	2023-10-31 17:04:18
27	36.0	23.0	2023-10-31 17:14:09
28	36.0	23.0	2023-10-31 17:14:15
29	36.0	23.0	2023-10-31 17:15:45
30	36.0	23.0	2023-10-31 17:19:33
31	74.0	25.0	2023-11-01 01:34:35

Figure 13: Show the records from database with query.

Command line to run server:



The screenshot shows a terminal window on a Raspberry Pi. The command `sudo python led_backend.py` has been executed. The output shows the server starting, serving the Flask app "led_backend" in production mode. It includes a warning to use a production WSGI server instead. The server is running on `http://0.0.0.0:80/`. It also shows the first two requests: a GET request for `/` and a GET request for `/favicon.ico`.

```
pi@raspberrypi:~/Desktop/web-server1 $ sudo python led_backend.py
* Serving Flask app "led_backend" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 321-268-469
b'Humidity = 37.00\r\n'
192.168.153.1 - - [01/Nov/2023 03:28:19] "GET / HTTP/1.1" 200 -
b'Temperature = 23.00\r\n'
192.168.153.1 - - [01/Nov/2023 03:28:19] "GET /favicon.ico HTTP/1.1" 200 -
```

Figure 14: Run server.

Explanation of C++ for Arduino board:

```
#include "dht.h" //include library

dht DHT;

#define DHT11_PIN 7 //set Pin 7 (LED)

int pin2 = 2; //set Pin 2 (LED)
int pin3 = 3; //set Pin 3 (LED)
int pin4 = 4; //set Pin 4 (LED)
int INA = 9; // set Pin 9(Fan motor)
int INB = 8; //set Pin 8(Fan motor)

int currentMode = 0; // default of mode is 0

void setup() {
    pinMode(pin2, OUTPUT);
    pinMode(pin3, OUTPUT);
    pinMode(pin4, OUTPUT);
    pinMode(INA, OUTPUT);
    pinMode(INB, OUTPUT);
    //Set default of Fan to low
    digitalWrite(INA, LOW);
    digitalWrite(INB, LOW);
    Serial.begin(9600);
}

void loop() {
    int value = Serial.read(); //Read the Serial input
    if (value == '1' && currentMode != 1) { // if the Serial input is
1 and current mode is not 1
        currentMode = 1; // set current mode = 1 (automatically)
    } else if ((value == '2' || value == '3') && currentMode != 2) {
        currentMode = 2; // set the current mode = 2 (For displaying
LED customly)
```

```

    } else if (value == '4' && currentMode != 4){
        currentMode = 4; // set current mode = 2 (For displaying LED
        customly and run fan motor)
    }
    int chk = DHT.read11(DHT11_PIN);
    Serial.print("Humidity = ");
    Serial.println(DHT.humidity); //Display humidity from sensor
    Serial.print("Temperature = ");
    Serial.println(DHT.temperature); //Display temperature from
    sensor
    if (currentMode == 1){ // the code will run automatically if
    currentMode is 1
        if (DHT.humidity >= 50 && DHT.humidity <= 60) {
            digitalWrite(pin2, HIGH); //trigger green light
            digitalWrite(pin3, LOW);
            digitalWrite(pin4, LOW);
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
        } else if (DHT.humidity >= 61 && DHT.humidity <= 70) {
            digitalWrite(pin2, LOW);
            digitalWrite(pin3, HIGH); //trigger yellow light
            digitalWrite(pin4, LOW);
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
        } else if (DHT.humidity >= 71 || DHT.humidity <= 49) {
            digitalWrite(pin2, LOW);
            digitalWrite(pin3, LOW);
            digitalWrite(pin4, HIGH); //trigger red light
            //run Fan motor clockwise
            digitalWrite(INA, HIGH);
            digitalWrite(INB, LOW);
        }
    }
}

```

```

    }
} else {
    if (value == '2') {
        digitalWrite(pin2, HIGH); //trigger green led
        digitalWrite(pin3, LOW);
        digitalWrite(pin4, LOW);
        digitalWrite(INA, LOW);
        digitalWrite(INB, LOW);
    } else if (value == '3') {
        digitalWrite(pin2, LOW);
        digitalWrite(pin3, HIGH); // trigger yellow led
        digitalWrite(pin4, LOW);
        digitalWrite(INA, LOW);
        digitalWrite(INB, LOW);
    } else if (value == '4') {
        digitalWrite(pin2, LOW);
        digitalWrite(pin3, LOW);
        digitalWrite(pin4, HIGH); //trigger red led
        digitalWrite(INA, HIGH); //trigger fan motor
        digitalWrite(INB, LOW);
    }
}

while (currentMode == 4) { // keep the fan run if current mode is 4
    digitalWrite(INA, HIGH);
    digitalWrite(INB, LOW);
}

Serial.print("CurrentMode = ");
Serial.println(currentMode); //print current mode for debugging.
delay(2000); // delay 2s
}

```

Explanation of Python code for web:

```

import serial
import MySQLdb
from flask import Flask, render_template

app = Flask(__name__)

sensors = { # Dictionary with the sensors
    1 : {'name' : 'Humidity', 'state' : 0 },
    2 : {'name' : 'Temperature', 'state' : 0 },
}

database = { # Dictionary with the database
    1 : {'name' : 'Database', 'state' : 0 },
    2 : [],
}

def read_sensor_data(): # Update the sensor data
    global sensors # Access the global variable sensors
    global database # Access the global variable database
    sensor = ser.readline() # Read the data from the Arduino
    print(sensor)
    sensor_str = sensor.decode('utf-8') # Decode the bytes to a string
    for line_sensor in sensor_str.split('\n'): # Split the string into lines
        if line_sensor.startswith('Humidity'): # Check if the line starts with
Humidity
            sensors[1]['state'] = float(line_sensor.split('=')[1].strip()) # Get
the value after the =
            elif line_sensor.startswith('Temperature'): # Check if the line starts
with Temperature
                sensors[2]['state'] = float(line_sensor.split('=')[1].strip()) # Get
the value after the =
def connect_to_database():
    try:
        return MySQLdb.connect("localhost", "pi", "", "sensor_data")
    except Exception as e:

```

```

        ser.write(b"4") # write serial input for Arduino code to trigger red
light
        print(f"Error connecting to the database: {e}")
        return None
# Main function when accessing the website
@app.route("/") # This is the main page
def index(): # This function will be executed when the main page is accessed
    read_sensor_data() # Update the sensor data
    # TODO: Read the status of the pins ON/OFF and update dictionary
    # This data will be sent to index.html (pins dictionary)
    templateData = { 'sensors' : sensors, 'database' :database } # Create a
dictionary with the data to be sent
    # Pass the template data into the template index.html and return it
    return render_template('index.html', **templateData) # Return the template

# Function with buttons to toggle to store the data into the database or not
@app.route("/<toggleDatabase>")
def toggle_store_data(toggleDatabase): # This function will be executed when the
main page is accessed
    read_sensor_data() # Update the sensor data
    dbConn = None
    try:
        dbConn = connect_to_database() # Connect to the database
        if dbConn:
            cursor = dbConn.cursor() # Create a cursor
            if toggleDatabase == "storeData": # Check if the button is storeData
                database[1]['state'] = 1 # Turn on the database
                ser.write(b"2") # write serial input for Arduino code to trigger
green light
                # Insert the data into the database
                cursor.execute("INSERT INTO sensor_data (humidity, temperature)
VALUES (%s, %s)",
                                (str(sensors[1]['state']),
str(sensors[2]['state'])))
                dbConn.commit() # Commit the changes

```

```

        print("Data stored in the database") # Print a message
        cursor.execute("SELECT * FROM sensor_data ORDER BY dataID DESC
LIMIT 10") # Get the last 10 data
        rows = cursor.fetchall() # Fetch the rows
        for row in rows: # Loop through the rows
            database[2].append({'humidity': row[1], 'temperature':
row[2], 'time_stamp': row[3]}) # Append the data to the database
        elif toggleDatabase == "noStoreData": # Check if the button is
noStoreData

            database[1]['state'] = 0 # Turn off the database
            ser.write(b"3") # write serial input for Arduino code to trigger
red light

        elif toggleDatabase == "getDatabase": # Check if the button is
getDatabase

            cursor.execute("SELECT * FROM sensor_data ORDER BY dataID DESC
LIMIT 10") # Get the last 10 data
            rows = cursor.fetchall()
            for row in rows:
                database[2].append({'humidity': row[1], 'temperature':
row[2], 'time_stamp': row[3]})
            finally:
                while len(database[2]) > 10: # Check if the database is greater than 10
                    database[2].pop(0) # Remove the first element
                if dbConn: # Close the database
                    dbConn.close()

            templateData = { 'sensors' : sensors, 'database' : database } # Create a
dictionary with the data to be sent
            return render_template('index.html', **templateData) # Return the template
@app.route("/automatically") #URL/automatically
def automatic():
    read_sensor_data() # Update the sensor data
    ser.write(b"1") #write serial input for Arduino code to trigger automatic
code

    templateData = { 'sensors' : sensors, 'database' : database }

```

```

    return render_template('index.html', **templateData)

#get data directly from arduino and send to the website
@app.route("/getData") #URL/getData
def getData():
    read_sensor_data() # Update the sensor data
    templateData = { 'sensors' : sensors, 'database' : database } # Create a
dictionary with the data to be sent
    return render_template('index.html', **templateData) # Return the template

# Main function when accessing the website
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyUSB0', 9600, timeout = 1) # Establish the
connection on a specific port
    ser.flush() # Clear the serial buffer
    app.run(host='0.0.0.0', port = 80, debug = True) # Run the app

```

Explanation of HTML code for display data:

```

<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
"></script>
    <meta name="author" content="Thanh Minh" />
    <meta name="description" content="SWE30011 - IOT Programming" />

    <title> Arduino Web Server </title>
    <style>
        table, th, td {

```



```

        border:1px solid black;
    }
</style>
</head>
<body class="container">
    <h1 class="text-center"> Arduino Web Server </h1>
    <p>
        <h2 class="my-4">
            Get the Data
        </h2>
        <!-- Display the buttons to get the data -->
        <a href="/getData" class="btn btn-info my-2">Get the current data</a>
        {% for sensor in sensors %} <!-- Loop through the sensors dictionary
-->
            <p>{{sensors[sensor]['name']}}: {{sensors[sensor]['state']}}</p>
        {% endfor %}
    </p>
    <p>
        <h2>Run Automatically</h2>
        <p>
            Note: Database won't be stored if it is automatically run
        </p>
        <a href="/automatically" class="btn btn-success">Run
Automatically</a> <!-- Display the button to run automatically -->
    </p>
    <p>
        <!-- Display the buttons to toggle the database -->
        <h3>{{database[1]['name']}}</h3>
        {% if database[1]['state'] == 1 %} <!-- Check if the database is on
or off -->
            is currently <strong>on</strong></h2>
        {% else %}
            is currently <strong>off</strong></h2>
        {% endif %}
        <br />

```

```

        <a href="/storeData" class="btn btn-primary mb-2">Store Data</a> <!--
Display the button to store data -->
        <br />
        <a href="/noStoreData" class="btn btn-secondary">No Store Data</a>
<!-- Display the button to no store data -->
    </p>
    <p>
        <h2>Data from Database</h2>
        <p>
            Note: Time may not the same with your local time due to server
configuration
            <br />
            Maximum display the last 10 record of data
            <br />
            You need to store the data in the database first so it can get
data from database
        </p>
        <a href="/getDatabase" class="btn btn-dark mb-4">Get data from
database</a> <!-- Display the button to get data from database -->
        <table class="table table-striped table-bordered">
            <th class="text-center" scope="col">Humidity</th>
            <th class="text-center" scope="col">Temperature</th>
            <th class="text-center" scope="col">Date and Time</th>
            {% for entry in database[2] %} <!-- Loop through the database
dictionary -->
                <tr>
                    <td class="text-center">{{ entry['humidity'] }}</td>
                    <td class="text-center">{{ entry['temperature'] }}</td>
                    <td class="text-center">{{ entry['time_stamp'] }}</td>
                </tr>
            {% endfor %}
        </table>

    </p>

```

```
<p>
  <h2>All Commands</h2> <!-- Display all the buttons -->
  <ol>
    <li class="my-1"><a href="/getData" class="btn btn-info">Get the
current data</a></li>
    <li class="my-1"><a href="/automatically" class="btn btn-
success">Run Automatically</a></li>
    <li class="my-1"><a href="/storeData" class="btn btn-
primary">Store Data</a></li>
    <li class="my-1"><a href="/noStoreData" class="btn btn-
secondary">No Store Data</a></li>
    <li class="my-1"><a href="/getDatabase" class="btn btn-dark mb-
4">Get data from database</a></li>
  </ol>
</p>
</body>
</html>
```

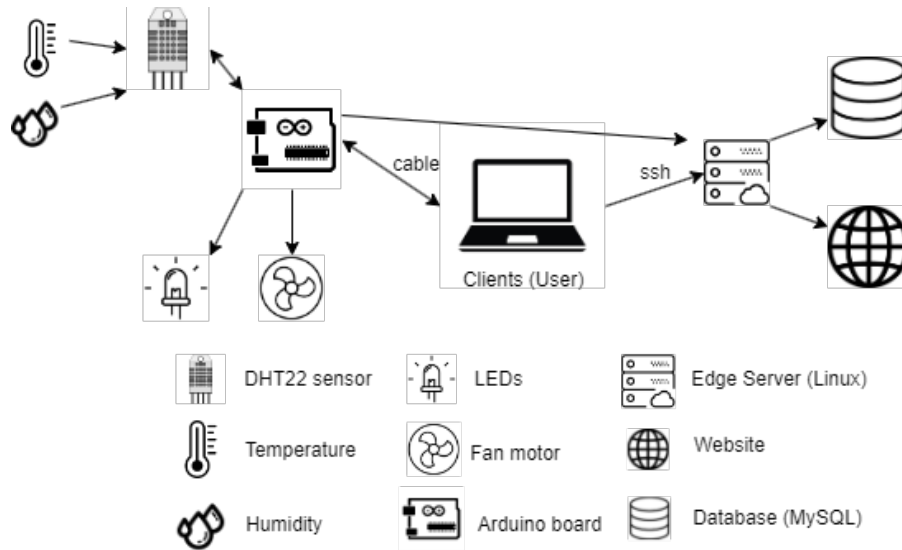
V. Resources

Circuit Basics (2023). *How to Use a DHT11 Humidity Sensor on the Arduino - Ultimate Guide to the Arduino #38. YouTube*. Available at: <https://www.youtube.com/watch?v=dJJAQxyryoQ> [Accessed 25 Oct. 2023].

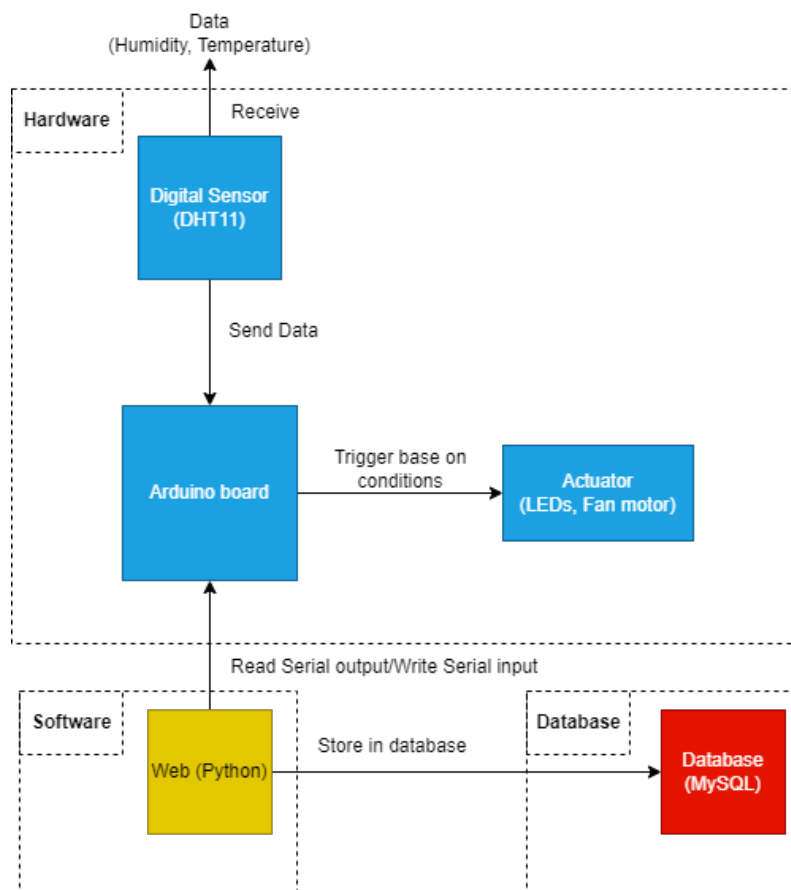
PLNTS.com. (2021). *Temperature and humidity*. [online] Available at: <https://plnts.com/en/care/doctor/temperature-and-humidity> [Accessed 28 Oct. 2023].

VI. Appendix

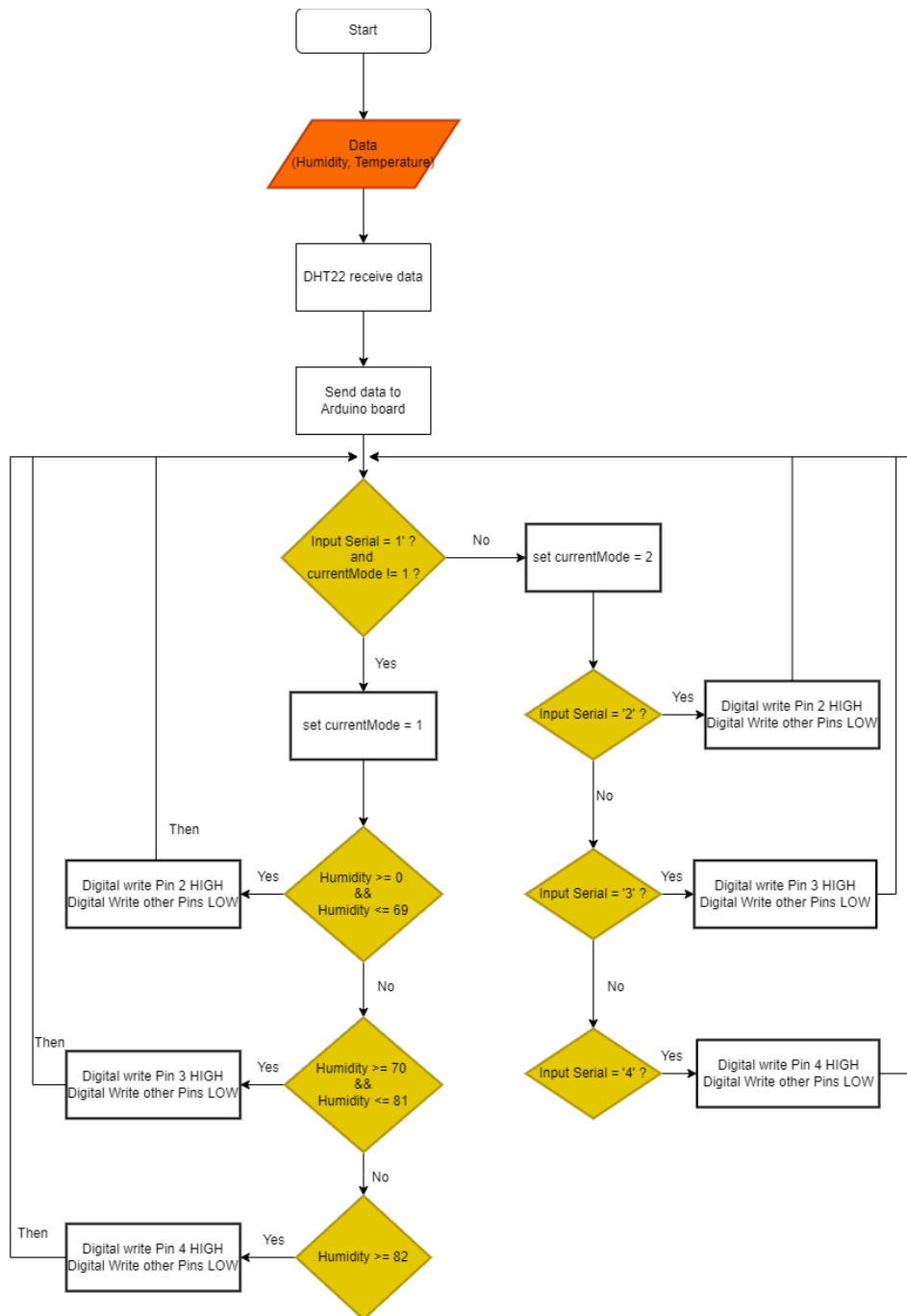
1. Sketch of proposed system



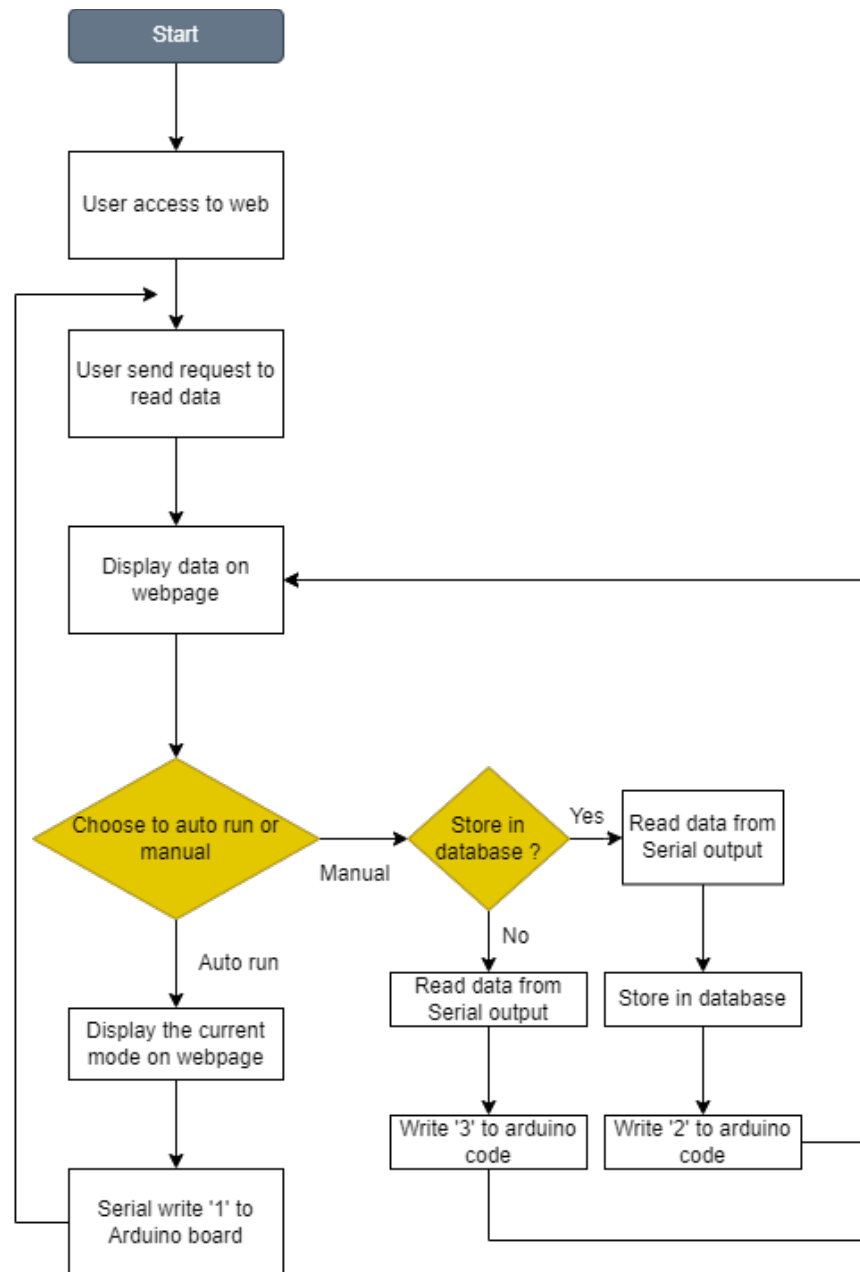
2. Block diagram for proposed system



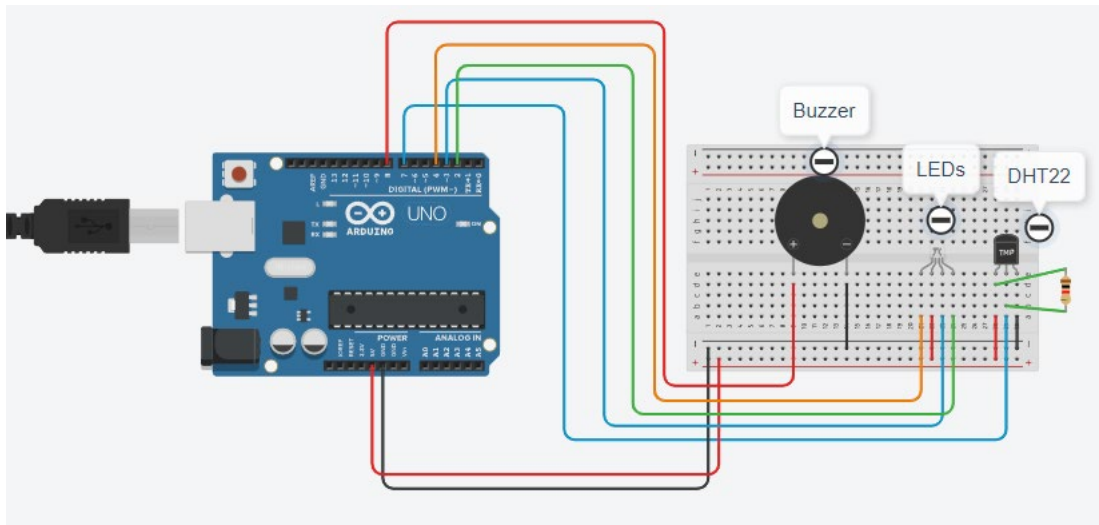
3. Flow chart for how Arduino system



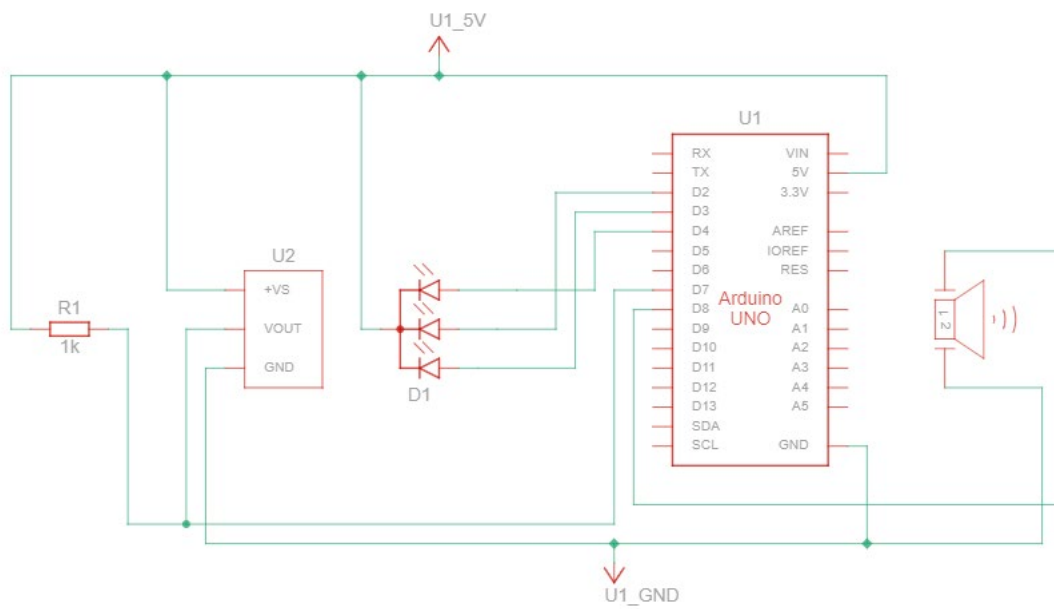
4. Flow chart for Python code and send request.



5. Tinkercad model of proposed system



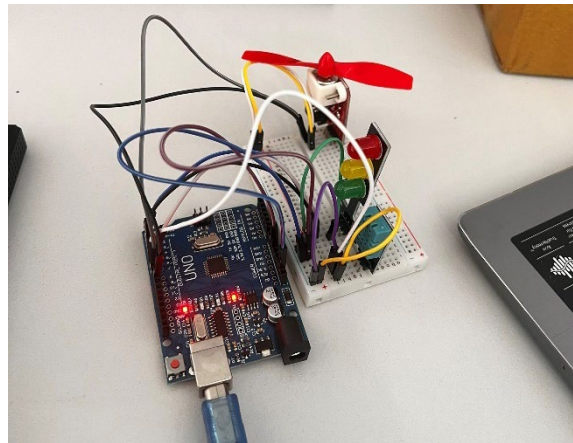
6. Schematic view of model Tinkercad



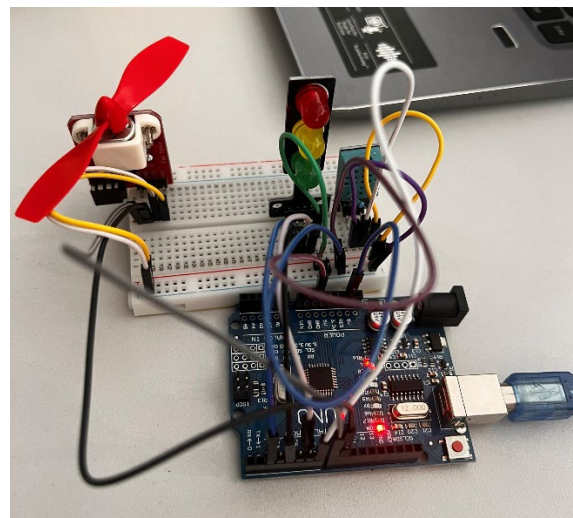
7. SQL query for creating table

```
CREATE TABLE sensor_data (  
  dataID INT AUTO_INCREMENT NOT NULL PRIMARY KEY,  
  humidity VARCHAR(10) NOT NULL,  
  temperature VARCHAR(10) NOT NULL,  
  time TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP  
);
```

8. Side view of physical system



9. Front view of physical system



10. Visual for getting data command

Arduino Web Server

Get the Data

Get the current data

Humidity: 73.0

Temperature: 25.0

11. Visual for automatically run command

Run Automatically

Note: Database won't be stored if it is automatically run

Run Automatically

12. Visual for store database command

Database

is currently **on**

Store Data

No Store Data

13. Visual for not storing database command

Database

is currently **off**

Store Data

No Store Data

14. Visual for displaying data from database

Data from Database

Note: Time may not the same with your local time due to server configuration

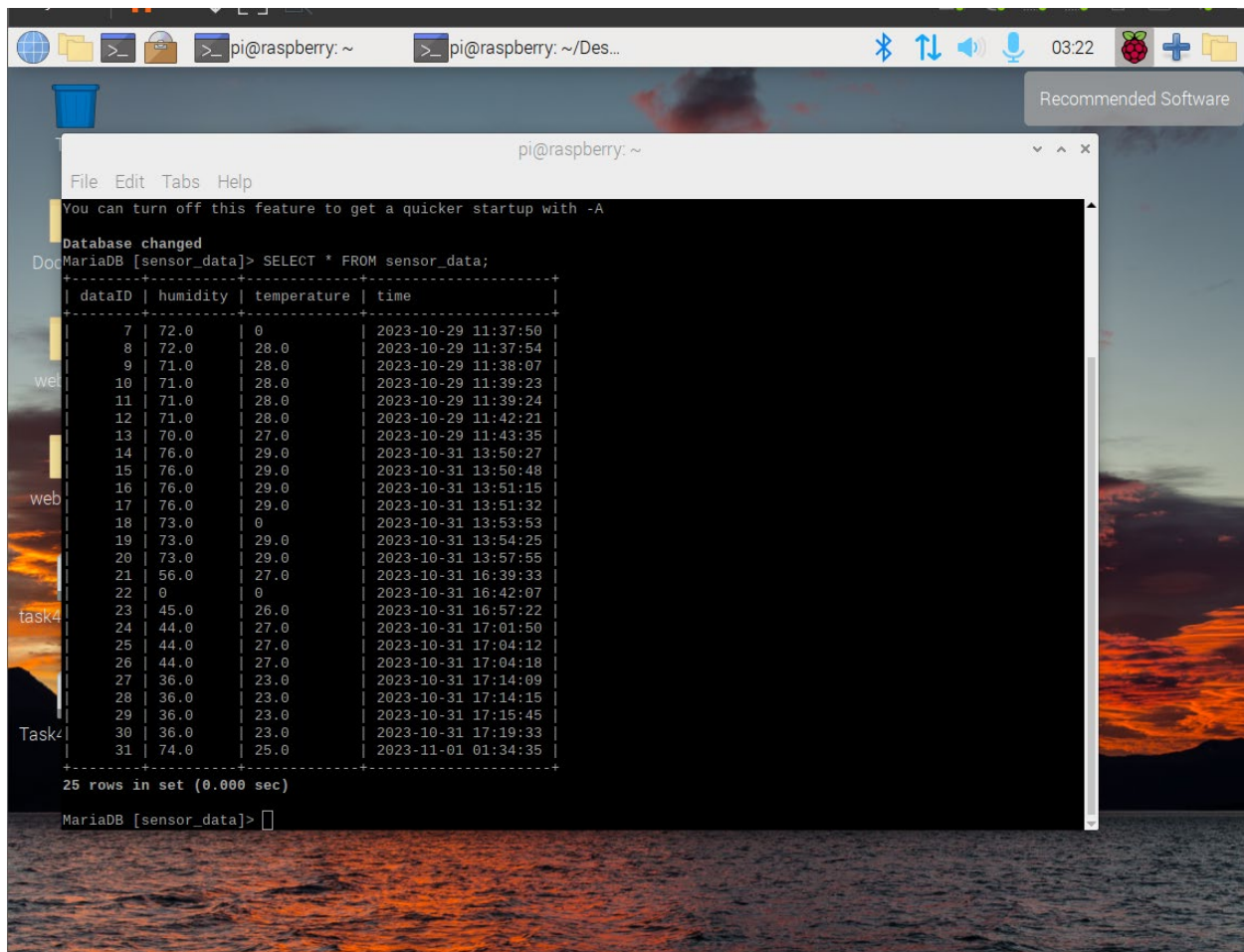
Maximum display the last 10 record of data

You need to store the data in the database first so it can get data from database

Get data from database

Humidity	Temperature	Date and Time
74.0	25.0	2023-11-01 01:34:35
36.0	23.0	2023-10-31 17:19:33
36.0	23.0	2023-10-31 17:15:45
36.0	23.0	2023-10-31 17:14:15
36.0	23.0	2023-10-31 17:14:09
44.0	27.0	2023-10-31 17:04:18
44.0	27.0	2023-10-31 17:04:12
44.0	27.0	2023-10-31 17:01:50
45.0	26.0	2023-10-31 16:57:22
0	0	2023-10-31 16:42:07

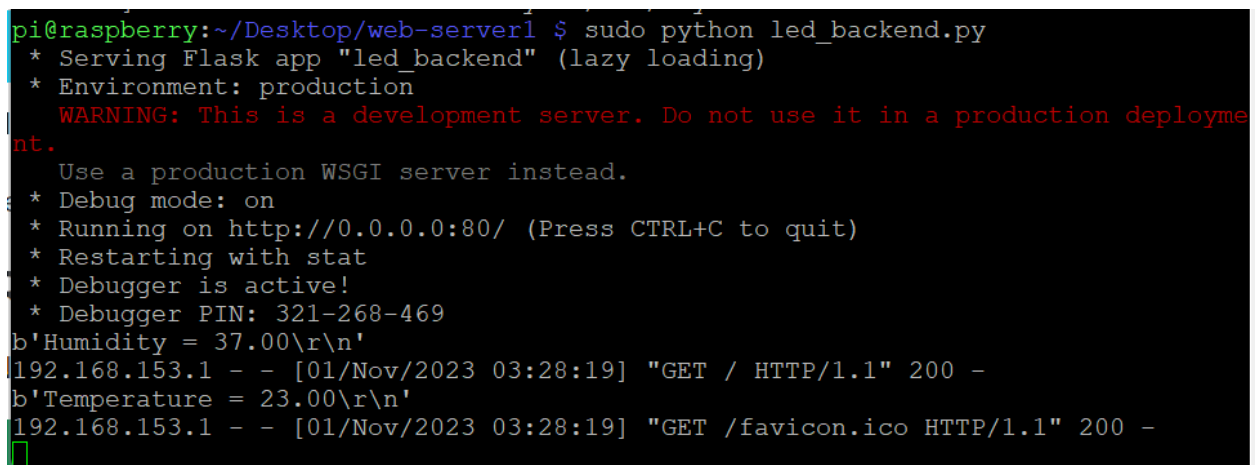
15. Display data from database in edge server



The screenshot shows a Raspberry Pi desktop with a terminal window open. The terminal displays the output of a SQL query executed in MariaDB. The query is `SELECT * FROM sensor_data;` and the result is a table with 31 rows. The table has four columns: `dataID`, `humidity`, `temperature`, and `time`. The data shows sensor readings from October 29 to November 1, 2023. The desktop background is a scenic image of a sunset over water. The terminal window title is `pi@raspberrypi: ~`.

```
pi@raspberrypi: ~  
File Edit Tabs Help  
You can turn off this feature to get a quicker startup with -A  
Database changed  
MariaDB [sensor_data]> SELECT * FROM sensor_data;  
+-----+-----+-----+-----+  
| dataID | humidity | temperature | time |  
+-----+-----+-----+-----+  
| 7 | 72.0 | 0 | 2023-10-29 11:37:50 |  
| 8 | 72.0 | 28.0 | 2023-10-29 11:37:54 |  
| 9 | 71.0 | 28.0 | 2023-10-29 11:38:07 |  
| 10 | 71.0 | 28.0 | 2023-10-29 11:39:23 |  
| 11 | 71.0 | 28.0 | 2023-10-29 11:39:24 |  
| 12 | 71.0 | 28.0 | 2023-10-29 11:42:21 |  
| 13 | 70.0 | 27.0 | 2023-10-29 11:43:35 |  
| 14 | 76.0 | 29.0 | 2023-10-31 13:50:27 |  
| 15 | 76.0 | 29.0 | 2023-10-31 13:50:48 |  
| 16 | 76.0 | 29.0 | 2023-10-31 13:51:15 |  
| 17 | 76.0 | 29.0 | 2023-10-31 13:51:32 |  
| 18 | 73.0 | 0 | 2023-10-31 13:53:53 |  
| 19 | 73.0 | 29.0 | 2023-10-31 13:54:25 |  
| 20 | 73.0 | 29.0 | 2023-10-31 13:57:55 |  
| 21 | 56.0 | 27.0 | 2023-10-31 16:39:33 |  
| 22 | 0 | 0 | 2023-10-31 16:42:07 |  
| 23 | 45.0 | 26.0 | 2023-10-31 16:57:22 |  
| 24 | 44.0 | 27.0 | 2023-10-31 17:01:50 |  
| 25 | 44.0 | 27.0 | 2023-10-31 17:04:12 |  
| 26 | 44.0 | 27.0 | 2023-10-31 17:04:18 |  
| 27 | 36.0 | 23.0 | 2023-10-31 17:14:09 |  
| 28 | 36.0 | 23.0 | 2023-10-31 17:14:15 |  
| 29 | 36.0 | 23.0 | 2023-10-31 17:15:45 |  
| 30 | 36.0 | 23.0 | 2023-10-31 17:19:33 |  
| 31 | 74.0 | 25.0 | 2023-11-01 01:34:35 |  
+-----+-----+-----+-----+  
25 rows in set (0.000 sec)  
MariaDB [sensor_data]>
```

16. Command line to run server



The screenshot shows a terminal window on a Raspberry Pi. The command `sudo python led_backend.py` has been executed. The output shows that the Flask application "led_backend" is running in production mode. A warning message states: "WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead." The server is running on `http://0.0.0.0:80/`. The output also shows the status of the server, including the debug mode, the running URL, and the restart command. The output ends with the status of the server, showing the humidity and temperature values.

```
pi@raspberrypi:~/Desktop/web-server1 $ sudo python led_backend.py  
* Serving Flask app "led_backend" (lazy loading)  
* Environment: production  
WARNING: This is a development server. Do not use it in a production deployment.  
Use a production WSGI server instead.  
* Debug mode: on  
* Running on http://0.0.0.0:80/ (Press CTRL+C to quit)  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 321-268-469  
b'Humidity = 37.00\r\n'  
192.168.153.1 - - [01/Nov/2023 03:28:19] "GET / HTTP/1.1" 200 -  
b'Temperature = 23.00\r\n'  
192.168.153.1 - - [01/Nov/2023 03:28:19] "GET /favicon.ico HTTP/1.1" 200 -  
█
```

17. C++ code for Arduino

```
#include "dht.h" //include library

dht DHT;

#define DHT11_PIN 7 //set Pin 7 (LED)

int pin2 = 2; //set Pin 2 (LED)
int pin3 = 3; //set Pin 3 (LED)
int pin4 = 4; //set Pin 4 (LED)
int INA = 9; // set Pin 9(Fan motor)
int INB = 8; //set Pin 8(Fan motor)

int currentMode = 0; // default of mode is 0

void setup() {
    pinMode(pin2, OUTPUT);
    pinMode(pin3, OUTPUT);
    pinMode(pin4, OUTPUT);
    pinMode(INA, OUTPUT);
    pinMode(INB, OUTPUT);
    //Set default of Fan to low
    digitalWrite(INA, LOW);
    digitalWrite(INB, LOW);
    Serial.begin(9600);
}

void loop() {
    int value = Serial.read(); //Read the Serial input
    if (value == '1' && currentMode != 1) { // if the Serial input is
1 and current mode is not 1
        currentMode = 1; // set current mode = 1 (automatically)
    } else if ((value == '2' || value == '3') && currentMode != 2) {
```

```

        currentMode = 2; // set the current mode = 2 (For displaying
        LED customly)
    } else if (value == '4' && currentMode != 4){
        currentMode = 4; // set current mode = 2 (For displaying LED
        customly and run fan motor)
    }
    int chk = DHT.read11(DHT11_PIN);
    Serial.print("Humidity = ");
    Serial.println(DHT.humidity); //Display humidity from sensor
    Serial.print("Temperature = ");
    Serial.println(DHT.temperature); //Display temperature from
    sensor
    if (currentMode == 1){ // the code will run automatically if
    currentMode is 1
        if (DHT.humidity >= 50 && DHT.humidity <= 60) {
            digitalWrite(pin2, HIGH); //trigger green light
            digitalWrite(pin3, LOW);
            digitalWrite(pin4, LOW);
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
        } else if (DHT.humidity >= 61 && DHT.humidity <= 70) {
            digitalWrite(pin2, LOW);
            digitalWrite(pin3, HIGH); //trigger yellow light
            digitalWrite(pin4, LOW);
            digitalWrite(INA, LOW);
            digitalWrite(INB, LOW);
        } else if (DHT.humidity >= 71 || DHT.humidity <= 49) {
            digitalWrite(pin2, LOW);
            digitalWrite(pin3, LOW);
            digitalWrite(pin4, HIGH); //trigger red light
            //run Fan motor clockwise

```

```

        digitalWrite(INA, HIGH);
        digitalWrite(INB, LOW);
    }
} else {
    if (value == '2') {
        digitalWrite(pin2, HIGH); //trigger green led
        digitalWrite(pin3, LOW);
        digitalWrite(pin4, LOW);
        digitalWrite(INA, LOW);
        digitalWrite(INB, LOW);
    } else if (value == '3') {
        digitalWrite(pin2, LOW);
        digitalWrite(pin3, HIGH); // trigger yellow led
        digitalWrite(pin4, LOW);
        digitalWrite(INA, LOW);
        digitalWrite(INB, LOW);
    } else if (value == '4') {
        digitalWrite(pin2, LOW);
        digitalWrite(pin3, LOW);
        digitalWrite(pin4, HIGH); //trigger red led
        digitalWrite(INA, HIGH); //trigger fan motor
        digitalWrite(INB, LOW);
    }
}

}

while (currentMode == 4) { // keep the fan run if current mode is 4
    digitalWrite(INA, HIGH);
    digitalWrite(INB, LOW);
}

Serial.print("CurrentMode = ");
Serial.println(currentMode); //print current mode for debugging.
delay(2000); // delay 2s

```

```
}
```

18. Python code

```
import serial
import MySQLdb
from flask import Flask, render_template

app = Flask(__name__)

sensors = { # Dictionary with the sensors
    1 : {'name' : 'Humidity', 'state' : 0 },
    2 : {'name' : 'Temperature', 'state' : 0 },
}

database = { # Dictionary with the database
    1 : {'name' : 'Database', 'state' : 0 },
    2 : [],
}

def read_sensor_data(): # Update the sensor data
    global sensors # Access the global variable sensors
    global database # Access the global variable database
    sensor = ser.readline() # Read the data from the Arduino
    print(sensor)
    sensor_str = sensor.decode('utf-8') # Decode the bytes to a string
    for line_sensor in sensor_str.split('\n'): # Split the string into lines
        if line_sensor.startswith('Humidity'): # Check if the line starts with
Humidity
            sensors[1]['state'] = float(line_sensor.split('=')[1].strip()) # Get
the value after the =
            elif line_sensor.startswith('Temperature'): # Check if the line starts
with Temperature
                sensors[2]['state'] = float(line_sensor.split('=')[1].strip()) # Get
the value after the =
def connect_to_database():
    try:
```



```

        return MySQLdb.connect("localhost", "pi", "", "sensor_data")
    except Exception as e:
        ser.write(b"4") # write serial input for Arduino code to trigger red
light
        print(f"Error connecting to the database: {e}")
        return None
# Main function when accessing the website
@app.route("/") # This is the main page
def index(): # This function will be executed when the main page is accessed
    read_sensor_data() # Update the sensor data
    # TODO: Read the status of the pins ON/OFF and update dictionary
    # This data will be sent to index.html (pins dictionary)
    templateData = { 'sensors' : sensors, 'database' :database } # Create a
dictionary with the data to be sent
    # Pass the template data into the template index.html and return it
    return render_template('index.html', **templateData) # Return the template

# Function with buttons to toggle to store the data into the database or not
@app.route("/<toggleDatabase>")
def toggle_store_data(toggleDatabase): # This function will be executed when the
main page is accessed
    read_sensor_data() # Update the sensor data
    dbConn = None
    try:
        dbConn = connect_to_database() # Connect to the database
        if dbConn:
            cursor = dbConn.cursor() # Create a cursor
            if toggleDatabase == "storeData": # Check if the button is storeData
                database[1]['state'] = 1 # Turn on the database
                ser.write(b"2") # write serial input for Arduino code to trigger
green light
                # Insert the data into the database
                cursor.execute("INSERT INTO sensor_data (humidity, temperature)
VALUES (%s, %s)",

```

```

        (str(sensors[1]['state']),
str(sensors[2]['state'])))
        dbConn.commit() # Commit the changes
        print("Data stored in the database") # Print a message
        cursor.execute("SELECT * FROM sensor_data ORDER BY dataID DESC
LIMIT 10") # Get the last 10 data
        rows = cursor.fetchall() # Fetch the rows
        for row in rows: # Loop through the rows
            database[2].append({'humidity': row[1], 'temperature':
row[2], 'time_stamp': row[3]}) # Append the data to the database
        elif toggleDatabase == "noStoreData": # Check if the button is
noStoreData
            database[1]['state'] = 0 # Turn off the database
            ser.write(b"3") # write serial input for Arduino code to trigger
red light

        elif toggleDatabase == "getDatabase": # Check if the button is
getDatabase
            cursor.execute("SELECT * FROM sensor_data ORDER BY dataID DESC
LIMIT 10") # Get the last 10 data
            rows = cursor.fetchall()
            for row in rows:
                database[2].append({'humidity': row[1], 'temperature':
row[2], 'time_stamp': row[3]})
        finally:
            while len(database[2]) > 10: # Check if the database is greater than 10
                database[2].pop(0) # Remove the first element
            if dbConn: # Close the database
                dbConn.close()
            templateData = { 'sensors' : sensors, 'database' : database } # Create a
dictionary with the data to be sent
            return render_template('index.html', **templateData) # Return the template
@app.route("/automatically") #URL/automatically
def automatic():
    read_sensor_data() # Update the sensor data

```

```

    ser.write(b"1") #write serial input for Arduino code to trigger automatic
code
    templateData = { 'sensors' : sensors, 'database' : database }
    return render_template('index.html', **templateData)

#get data directly from arduino and send to the website
@app.route("/getData") #URL/getData
def getData():
    read_sensor_data() # Update the sensor data
    templateData = { 'sensors' : sensors, 'database' : database } # Create a
dictionary with the data to be sent
    return render_template('index.html', **templateData) # Return the template

# Main function when accessing the website
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyUSB0', 9600, timeout = 1) # Establish the
connection on a specific port
    ser.flush() # Clear the serial buffer
    app.run(host='0.0.0.0', port = 80, debug = True) # Run the app

```

19. HTML code

```

<!DOCTYPE html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
"></script>
    <meta name="author" content="Thanh Minh" />
    <meta name="description" content="SWE30011 - IOT Programming" />

```

```

<title> Arduino Web Server </title>
<style>
    table, th, td {
        border:1px solid black;
    }
</style>
</head>
<body class="container">
    <h1 class="text-center"> Arduino Web Server </h1>
    <p>
        <h2 class="my-4">
            Get the Data
        </h2>
        <!-- Display the buttons to get the data -->
        <a href="/getData" class="btn btn-info my-2">Get the current data</a>
        {% for sensor in sensors %} <!-- Loop through the sensors dictionary
-->
            <p>{{sensors[sensor]['name']}}: {{sensors[sensor]['state']}}</p>
        {% endfor %}
    </p>
    <p>
        <h2>Run Automatically</h2>
        <p>
            Note: Database won't be stored if it is automatically run
        </p>
        <a href="/automatically" class="btn btn-success">Run
Automatically</a> <!-- Display the button to run automatically -->
    </p>
    <p>
        <!-- Display the buttons to toggle the database -->
        <h3>{{database[1]['name']}}</h3>
        {% if database[1]['state'] == 1 %} <!-- Check if the database is on
or off -->
            is currently <strong>on</strong></h2>
        {% else %}

```

```

        is currently <strong>off</strong></h2>
    {% endif %}
    <br />
    <a href="/storeData" class="btn btn-primary mb-2">Store Data</a> <!--
Display the button to store data -->
    <br />
    <a href="/noStoreData" class="btn btn-secondary">No Store Data</a>
<!-- Display the button to no store data -->
</p>
<p>
    <h2>Data from Database</h2>
    <p>
        Note: Time may not the same with your local time due to server
configuration
    <br />
        Maximum display the last 10 record of data
    <br />
        You need to store the data in the database first so it can get
data from database
    </p>
    <a href="/getDatabase" class="btn btn-dark mb-4">Get data from
database</a> <!-- Display the button to get data from database -->
    <table class="table table-striped table-bordered">
        <th class="text-center" scope="col">Humidity</th>
        <th class="text-center" scope="col">Temperature</th>
        <th class="text-center" scope="col">Date and Time</th>
        {% for entry in database[2] %} <!-- Loop through the database
dictionary -->
        <tr>
            <td class="text-center">{{ entry['humidity'] }}</td>
            <td class="text-center">{{ entry['temperature'] }}</td>
            <td class="text-center">{{ entry['time_stamp'] }}</td>
        </tr>
        {% endfor %}
    </table>

```

```
</p>

<p>
  <h2>All Commands</h2> <!-- Display all the buttons -->
  <ol>
    <li class="my-1"><a href="/getData" class="btn btn-info">Get the
current data</a></li>
    <li class="my-1"><a href="/automatically" class="btn btn-
success">Run Automatically</a></li>
    <li class="my-1"><a href="/storeData" class="btn btn-
primary">Store Data</a></li>
    <li class="my-1"><a href="/noStoreData" class="btn btn-
secondary">No Store Data</a></li>
    <li class="my-1"><a href="/getDatabase" class="btn btn-dark mb-
4">Get data from database</a></li>
  </ol>
</p>
</body>
</html>
```