SCHOOL OF COMPUTING
UUM COLLEGE OF ARTS AND SCIENCES

STTHK3113 SENSOR-BASED SYSTEMS
SEMESTER 6 (A242)

MIDTERM EXAM : TEMPERATURE AND HUMIDITY MONITORING WITH RELAY TRIGGER AND NEAR REAL-TIME GRAPH

LECTURED BY :
AHMAD HANIS BIN MOHD SHABLI

PREPARED BY

NAME	MATRIC NO.
GERALD TAN HOCK SOON	294879

SUBMISSION DATE : 30TH MAY 2025

Github link : https://github.com/Tacozn/Temperature-Humidity-Monitoring.git

Youtube link : https://youtu.be/ng4il1UtUH0

1.0 INTRODUCTION

This project implements a complete sensor-based system using an ESP32 microcontroller, a DHT11 sensor, a relay module, and a mobile application. The main objective is to collect temperature and humidity data every 10 seconds, store it in a MySQL database, trigger a relay based on configurable thresholds, and visualize the readings in near real-time using a Flutter mobile app. This solution is ideal for monitoring environments such as server rooms, greenhouses, or homes.

2.0 SYSTEM ARCHITECTURE

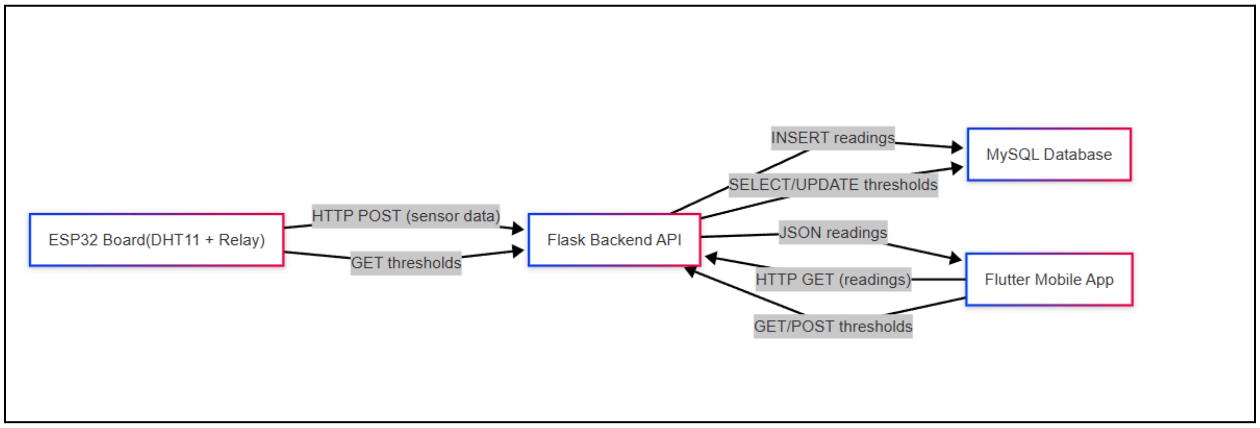


Figure 1: Overall system architecture showing data flow between the ESP32, Flask backend, MySQL database, and Flutter mobile app.

3.0 HARDWARE SETUP

3.1 Components Used:

- ESP32 Dev Board
- DHT11 Temperature & Humidity Sensor
- 1-Channel Relay Module

- ESP32 Training Board and jumper wires
- Type-C USB cable for power and upload

3.2 Wiring:

Component	**ESP32 Pin**
DHT11 VCC	3.3V
DHT11 GND	GND
DHT11 DATA	GPIO 4
Relay VCC	3.3V
Relay GND	GND
Relay IN	GPIO 5

Table 1: Pin configuration for DHT11 sensor and relay module with the ESP32.

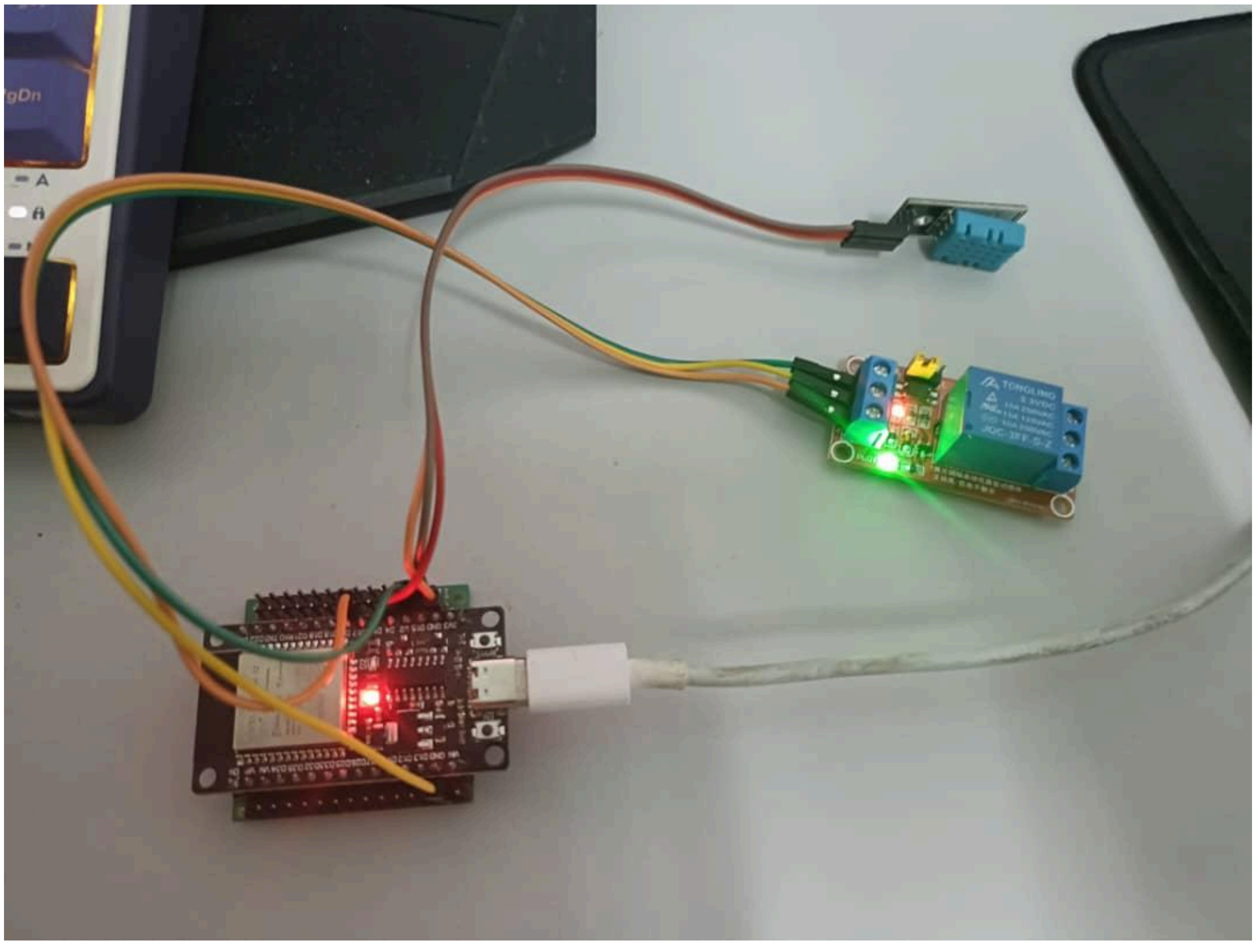


Figure 2: Hardware assembly of the ESP32, DHT11, and relay module on an ESP32 training board.

4.0 SETUP STEPS

The following steps were followed to set up and integrate the hardware, backend server, and mobile application:

4.1 Hardware Assembly

- The DHT11 sensor and relay were connected to the ESP32 using jumper wires on a breadboard.
- GPIO 4 was used for the DHT11 data pin, and GPIO 5 controlled the relay.
- Both modules were powered using the 3.3V and GND pins on the ESP32, with care taken to ensure secure connections.

4.2 ESP32 Firmware Upload

- The Arduino IDE was used to program the ESP32.
- Required libraries (DHT, WiFi, HTTPClient, and ArduinoJson) were installed.
- Wi-Fi credentials and backend IP address were configured in the code.
- The code was uploaded via USB, and Serial Monitor was used for debugging.

4.3 Backend Setup (Flask + MySQL)

- Python 3 and MySQL were installed locally.
- Required packages were installed using pip: flask, flask-cors, and mysql-connector-python.
- A MySQL database named sensor_db was created with two tables: readings and thresholds.
- The Flask app (app.py) was run to start the API server, listening on port 5000.

4.4 Mobile App (Flutter) Setup

- A new Flutter project was created.
- Dependencies (http, fl_chart) were added in pubspec.yaml.
- The UI was developed to display readings, relay status, and allow threshold updates.
- The app was tested using flutter run on a real Android device connected to the same Wi-Fi network.

4.5 Testing & Integration

- The ESP32 was tested to ensure it could read sensor values, fetch thresholds, and control the relay correctly.

- The mobile app was connected to the backend to verify real-time updates and threshold control.

- Threshold updates were tested end-to-end to confirm dynamic response by the ESP32.

5.0 SOFTWARE IMPLEMENTATION

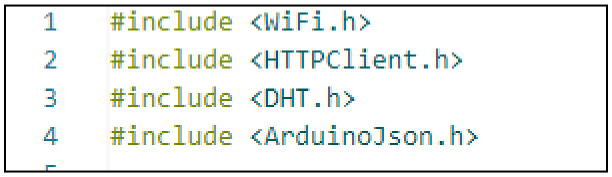
The system is composed of three core components: the ESP32 firmware, a backend server using Flask and MySQL, and a Flutter mobile application. Each layer plays a critical role in ensuring the system performs real-time monitoring, control, and visualization of temperature and humidity data.

5.1 ESP32 Firmware (Arduino)

The ESP32 acts as the main controller and is programmed using the Arduino framework. It connects to Wi-Fi and reads data from a DHT11 sensor every 10 seconds. Before each data transmission, the ESP32 sends an HTTP GET request to the backend to retrieve the latest user-defined temperature and humidity thresholds. This ensures that the relay is always triggered based on the most recent configuration.

If the measured temperature or humidity exceeds the thresholds, the ESP32 activates a relay module connected to GPIO 5. A JSON payload containing the current temperature, humidity, relay state, and timestamp is then sent via HTTP POST to the backend. The use of the HTTPClient and ArduinoJson libraries made it easier to structure and send the data. This setup enables accurate environmental control and seamless integration with the backend server.

Key Libraries Used:

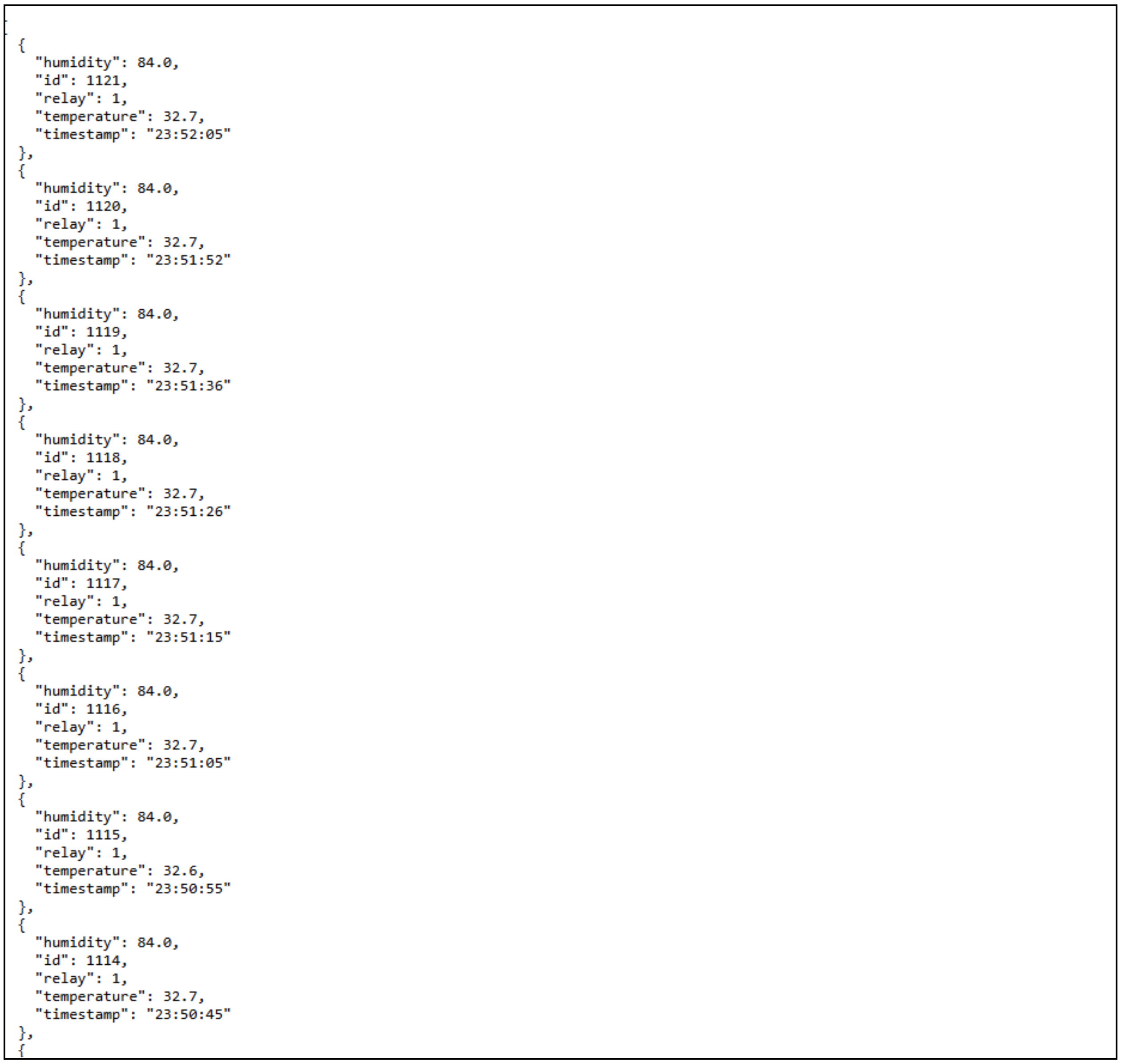


```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
#include <ArduinoJson.h>
```

5.2 Backend (Flask + MySQL)

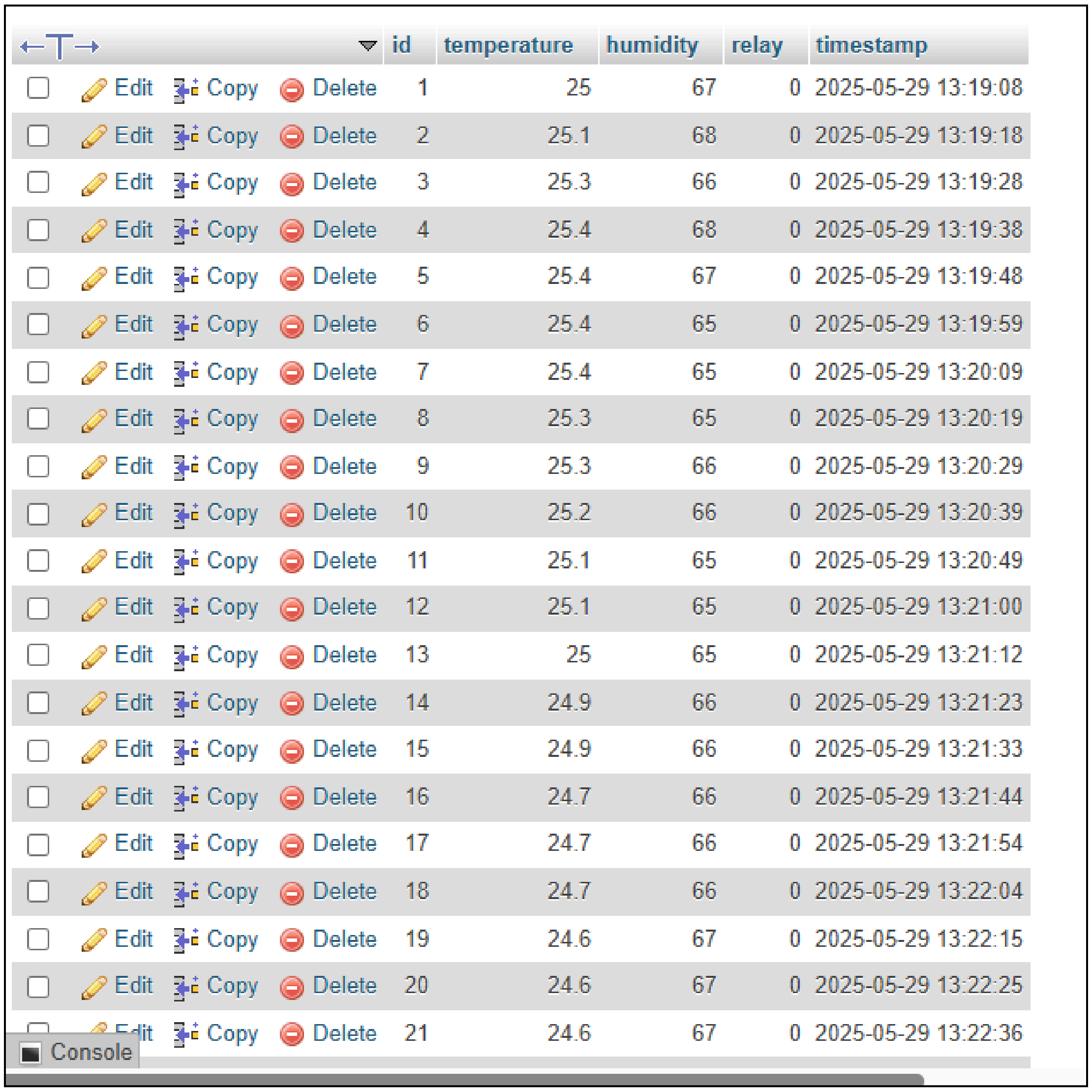
The backend server is built using Python's Flask framework and communicates with a MySQL database. It provides API endpoints that allow the ESP32 to send sensor data and retrieve threshold settings. It also allows the mobile app to retrieve recent data for visualization and update threshold values.

The database schema includes two tables: readings, which stores all temperature, humidity, relay, and timestamp values; and thresholds, which stores the current temperature and humidity thresholds. On initialization, the backend ensures that default thresholds are inserted if none exist. The API exposes four main routes: POST /api/data to receive sensor readings, GET /api/data to fetch recent readings, GET /api/thresholds to get the current thresholds, and POST /api/thresholds to update them. These endpoints are lightweight, efficient, and secured using CORS for safe communication with the Flutter app.



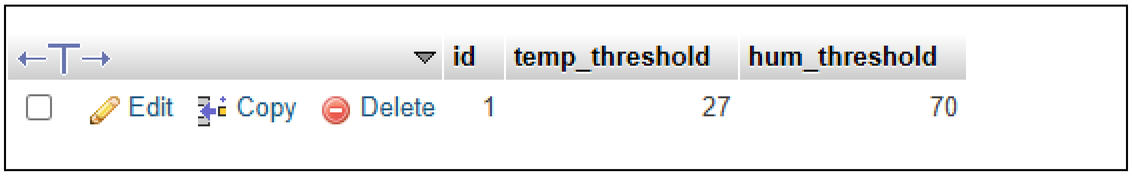
```
 {
   "humidity": 84.0,
   "id": 1121,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:52:05"
 },
 {
   "humidity": 84.0,
   "id": 1120,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:51:52"
 },
 {
   "humidity": 84.0,
   "id": 1119,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:51:36"
 },
 {
   "humidity": 84.0,
   "id": 1118,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:51:26"
 },
 {
   "humidity": 84.0,
   "id": 1117,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:51:15"
 },
 {
   "humidity": 84.0,
   "id": 1116,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:51:05"
 },
 {
   "humidity": 84.0,
   "id": 1115,
   "relay": 1,
   "temperature": 32.6,
   "timestamp": "23:50:55"
 },
 {
   "humidity": 84.0,
   "id": 1114,
   "relay": 1,
   "temperature": 32.7,
   "timestamp": "23:50:45"
 },
 {
```

Figure 3: Flask API endpoint showing JSON-formatted sensor data including temperature, humidity, relay status, and timestamp.



id	temperature	humidity	relay	timestamp
1	25	67	0	2025-05-29 13:19:08
2	25.1	68	0	2025-05-29 13:19:18
3	25.3	66	0	2025-05-29 13:19:28
4	25.4	68	0	2025-05-29 13:19:38
5	25.4	67	0	2025-05-29 13:19:48
6	25.4	65	0	2025-05-29 13:19:59
7	25.4	65	0	2025-05-29 13:20:09
8	25.3	65	0	2025-05-29 13:20:19
9	25.3	66	0	2025-05-29 13:20:29
10	25.2	66	0	2025-05-29 13:20:39
11	25.1	65	0	2025-05-29 13:20:49
12	25.1	65	0	2025-05-29 13:21:00
13	25	65	0	2025-05-29 13:21:12
14	24.9	66	0	2025-05-29 13:21:23
15	24.9	66	0	2025-05-29 13:21:33
16	24.7	66	0	2025-05-29 13:21:44
17	24.7	66	0	2025-05-29 13:21:54
18	24.7	66	0	2025-05-29 13:22:04
19	24.6	67	0	2025-05-29 13:22:15
20	24.6	67	0	2025-05-29 13:22:25
21	24.6	67	0	2025-05-29 13:22:36

Table 2: MySQL readings table displaying stored temperature, humidity, relay status, and timestamp entries.



id	temp_threshold	hum_threshold
1	27	70

Table 3: MySQL thresholds table storing current temperature and humidity thresholds used by the ESP32.

5.3 Mobile App (Flutter)

The mobile application is built using Flutter and serves as the user interface for monitoring the system. Upon launch, the app fetches both the latest sensor readings and the current threshold settings from the backend. It displays the current temperature, humidity, and relay status in a visually appealing dashboard, alongside dynamic line graphs generated using the `fl_chart` package.

A key feature of the app is its threshold configuration panel. Users can easily enter new threshold values, which are then sent to the backend via a POST request. Once updated, the new values are reflected both in the database and on the ESP32 during its next fetch cycle. The UI is built for responsiveness and clarity, with loading indicators, refresh controls, and visual feedback for success or error states. The app also includes auto-refresh every 30 seconds and manual refresh options to ensure data stays current without excessive user input.

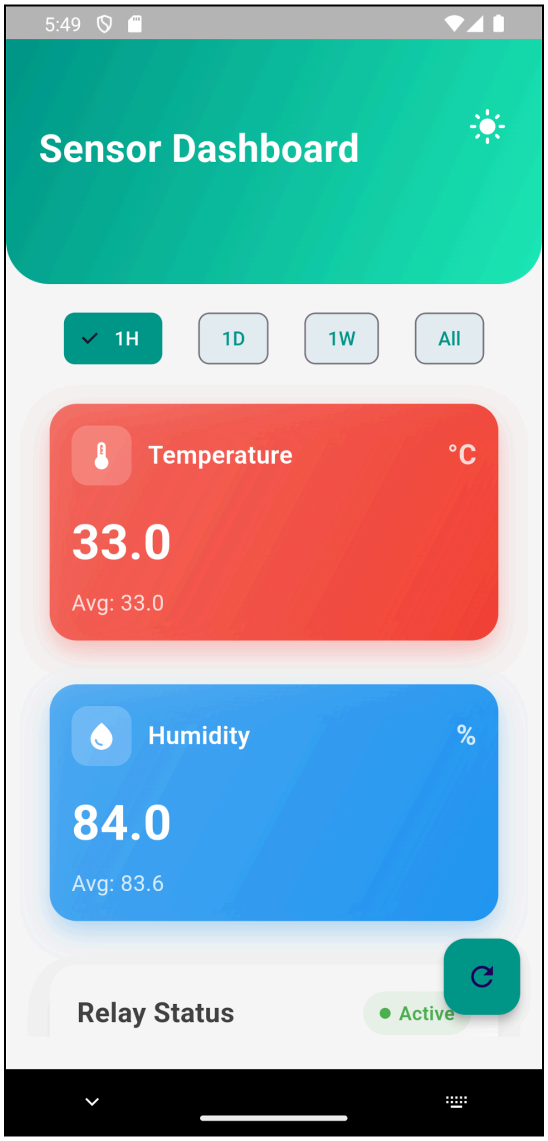
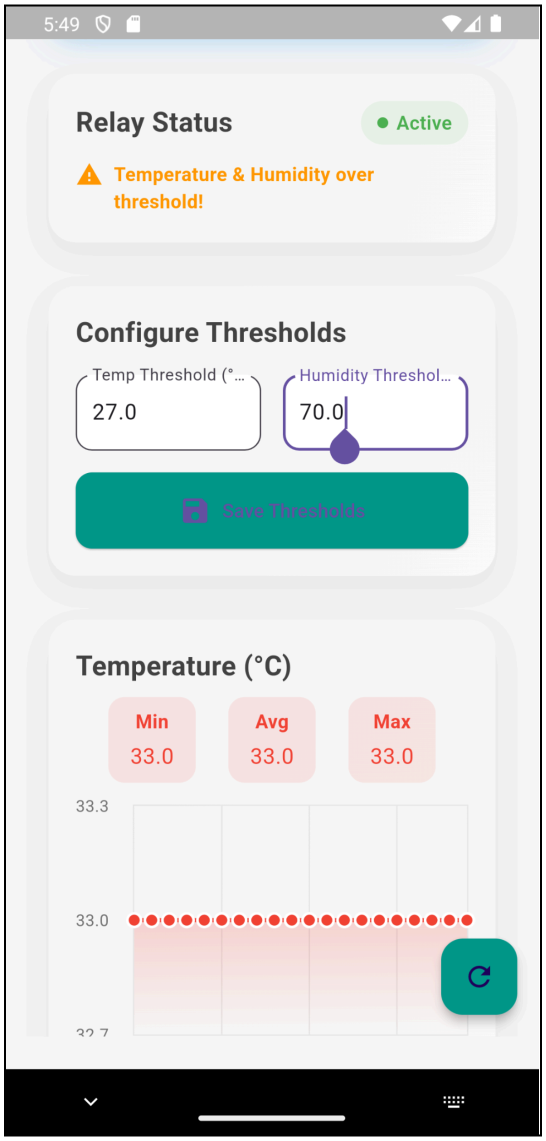


Figure 4: Flutter mobile app displaying current temperature and humidity readings in a clean, user-friendly interface.



5:49

Relay Status

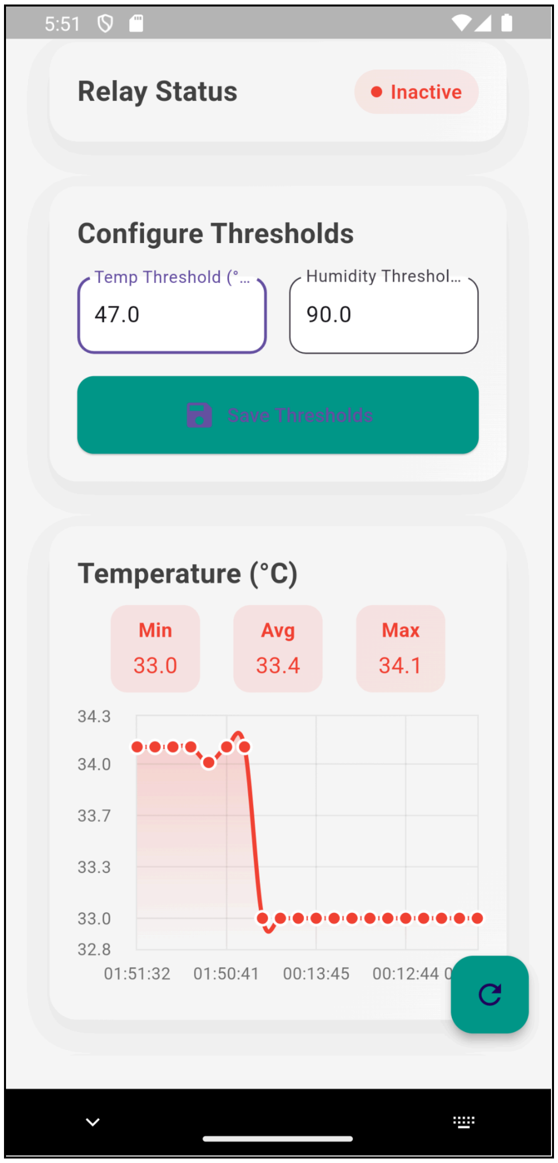
● Active

Temperature & Humidity over threshold!

Configure Thresholds

Temp Threshold (°…	Humidity Threshol…
27.0	70.0

Figure 5: App UI showing relay status as "Active" when both temperature and humidity exceed their respective thresholds.


Relay Status

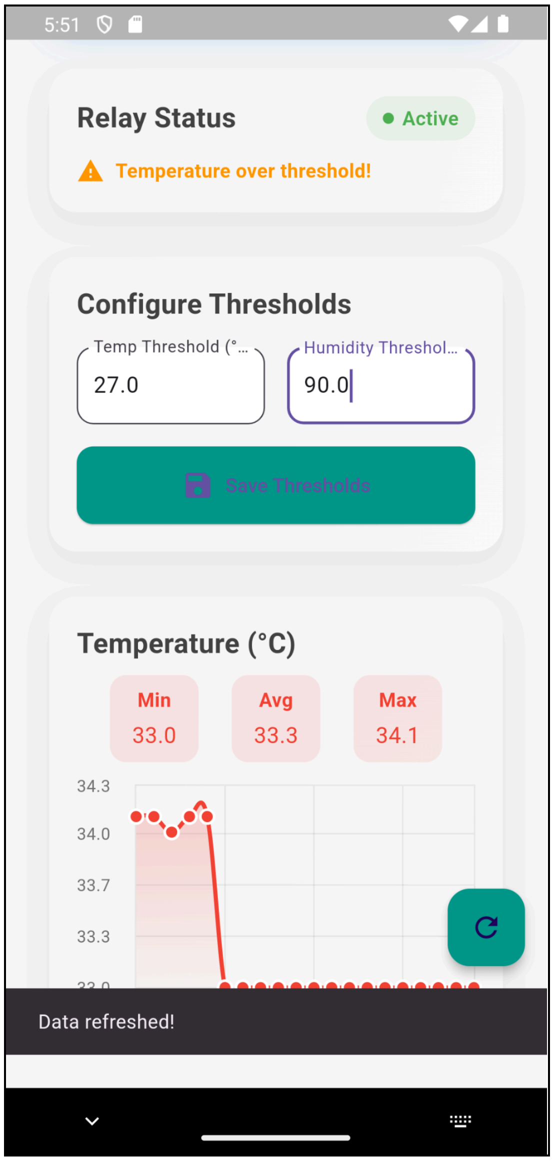
● Inactive

Configure Thresholds

Temp Threshold (°…: 47.0

Humidity Threshol…: 90.0

Figure 6: App UI showing relay status as "Inactive" when readings are below the configured thresholds.


5:51

Relay Status

Active

Temperature over threshold!

Configure Thresholds

Temp Threshold (°…

27.0

Humidity Threshol…

90.0

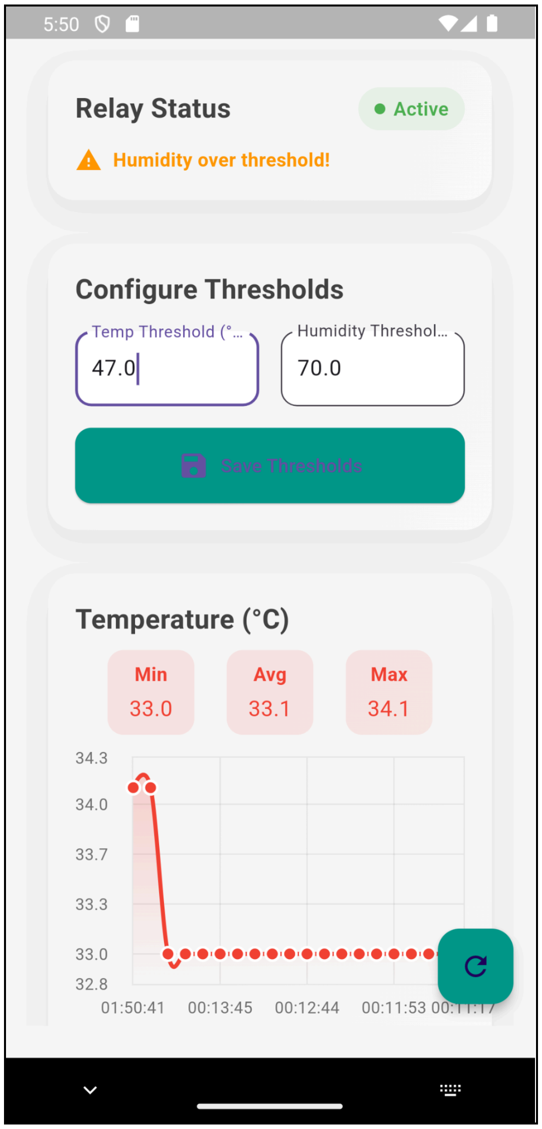
Save Thresholds

Temperature (°C)

Min	Avg	Max
33.0	33.3	34.1

Data refreshed!

Figure 7: Relay activated due to high temperature while humidity remains normal.


5:50

Relay Status ● Active

⚠ Humidity over threshold!

Configure Thresholds

Temp Threshold (°… 47.0

Humidity Threshol… 70.0

Temperature (°C)

Min	Avg	Max
33.0	33.1	34.1

Figure 8: Relay activated due to high humidity even though temperature is within limits.

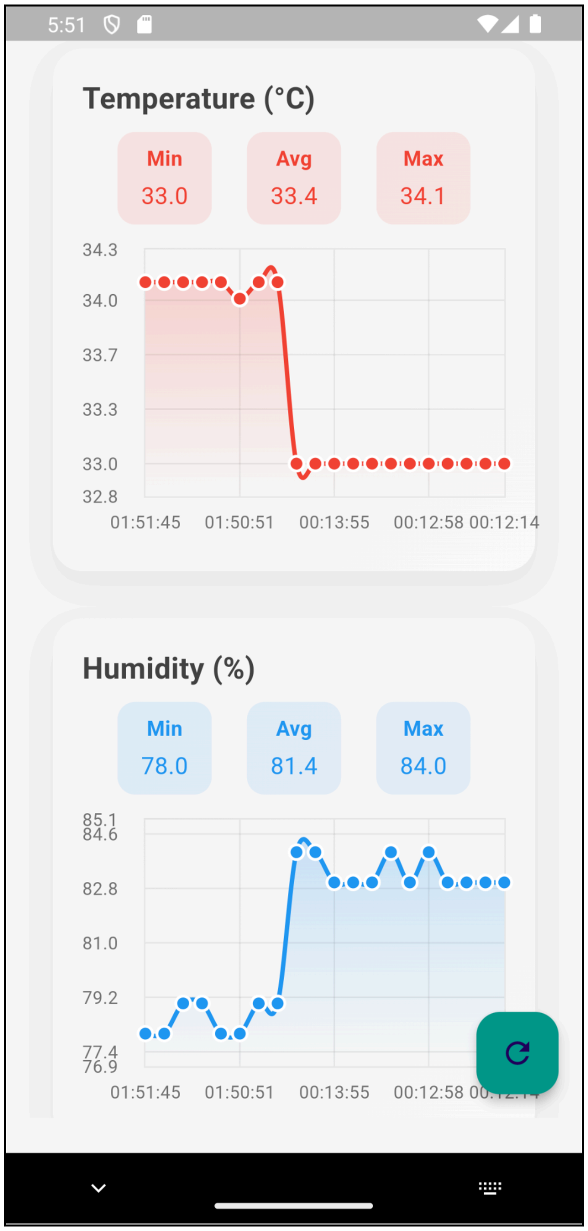


Figure 9: Line graphs in the mobile app showing temperature and humidity variations over time, including min, max, and average values.

6.0 CHALLENGES AND SOLUTIONS

Throughout the development of this project, several challenges emerged, both in hardware and software. One of the first issues encountered was that the DHT11 sensor occasionally returned “NaN” (Not a Number) values during readings. This was resolved by ensuring proper wiring and adding a short delay after sensor initialization to allow it to stabilize.

Another challenge was the Wi-Fi connection on the ESP32 occasionally failing or taking too long to reconnect after power loss. This was mitigated by implementing a basic reconnection loop in the setup() function, ensuring the ESP32 remained reliably connected to the network before proceeding to fetch thresholds or send data.

Lastly, getting the mobile app to reflect real-time data and threshold changes required refining the app’s state management and timing. Introducing a 30-second auto-refresh and a manual refresh button helped maintain a balance between responsiveness and performance, and all API errors are now handled gracefully with visual feedback.

7.0 IMPROVEMENTS

To enhance the system further, user authentication could be added to restrict access to threshold settings. Integrating a cloud-based database would allow remote monitoring from anywhere, instead of being limited to the local network.

The mobile app could also be expanded to include historical data views, enabling users to track trends over time. Adding more sensors, such as for air quality or soil moisture, would make the system more versatile. Lastly, push notifications could alert users when thresholds are exceeded, improving its usefulness in real-time applications.