Web-of-Things: A Mini Weather Station IoT using Sense HAT and MQTT

Group 4: [Sanzida Akter Chadne, Mst Jannatul Ferdows]

May 25, 2025

Abstract

This project explores how to build a small weather station using IoT technologies. We use a Raspberry Pi with a Sense HAT to collect data on temperature, humidity, and pressure. The goal is to create a system that can send this data in real time using MQTT, a lightweight messaging protocol. We also use a Flask web server to make the data available through a simple web interface. Our research questions focus on how well MQTT and RESTful APIs can work together in a basic IoT setup, and how such a low-cost system compares to existing commercial weather monitoring solutions. We show that even with simple tools, it is possible to build a reliable and useful system for collecting and sharing environmental data. This work adds to the understanding of how IoT systems can be designed to handle time-series data and support web-based access.

I. Introduction

The Internet of Things (IoT) is a growing field that connects physical objects—often called smart objects—to the internet. These devices combine sensing, computing, and communication capabilities to monitor and react to their environment. In this project, we explore how a basic IoT weather station can be built using affordable and accessible components, while following the principles of modern IoT architecture.

Our weather station is built using a Raspberry Pi and a Sense HAT. This setup allows us to collect live data on temperature, humidity, and atmospheric pressure. We transmit this data using MQTT, a lightweight publish/subscribe protocol well-suited for IoT due to its efficiency and low overhead. A Flask web server acts as the front end, offering RESTful access to the data and basic analytical functions.

This project directly engages with the course topics, especially in areas such as network protocols, system design, and real-time data handling. From the lectures, we learned that IoT systems must manage constrained resources, ensure reliable communication, and support modularity. Our implementation reflects these ideas through the use of microcontrollers, sensor integration, and message-oriented middleware.

We also compare our system with commercial weather stations to understand where a simple, open-source approach stands in terms of performance, cost, and extensibility. Our goal is to demonstrate how key IoT concepts like device communication, data flow, and real-time accessibility can be achieved with minimal hardware, in line with the layered IoT reference models introduced in lecture.

Through this project, we aim to gain practical insights into the design and deployment of smart, connected products—moving from theoretical understanding to hands-on implementation.

II. Background

Understanding the core technologies and architecture of IoT is essential to building reliable and scalable systems. We have already mentioned that this project is based on a practical implementation of a weather station using common IoT components and protocols. Below, we detail the key hardware and software technologies used, and relate them to established concepts from the course.

A. Sense HAT and Raspberry Pi

The hardware core of our weather station consists of a Raspberry Pi 3 Model B and the Sense HAT extension board. The Sense HAT provides integrated sensors for temperature, humidity, and pressure, making it ideal for environmental data monitoring. It communicates with the Raspberry Pi via the I2C interface, a common synchronous serial protocol in IoT devices. Python libraries offered by the Raspberry Pi Foundation simplify access to sensor readings, supporting rapid prototyping and data acquisition.

From a system architecture perspective, the Raspberry Pi acts as an IoT edge device. It combines sensing, local processing, and communication. It fits within the broader vision of smart objects and everyday items enhanced with computation and connectivity capabilities.

B. IoT Architecture and Protocol Stack

This project fits into the standard IoT architecture as described in ITU-T Y.2060, encompassing perception, network, and application layers. At the perception layer, the sensors gather environmental data. At the network layer, MQTT enables efficient communication. The application layer is represented by a Flask-based REST API, which allows users to access and interact with the system.

The layered design also follows the end-to-end principle of internet architecture. We use TCP/IP for reliable connectivity, with MQTT operating over TCP, ensuring message delivery integrity.

C. MQTT Middleware and Publish/Subscribe Model

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol specifically designed for constrained devices and low-bandwidth networks. It uses a publish-subscribe model that decouples data producers (publishers) from consumers (subscribers), making the system scalable and resilient to changes.

MQTT's strength lies in its ability to provide space, time, and synchronization decoupling. Our implementation uses the Paho MQTT client in Python. The broker routes messages between the Raspberry Pi (publisher) and subscribers, which helps log, display, or analyze the data. This perfectly aligns with the telemetry pattern where devices continuously push status updates or measurements to subscribers.

D. Time-Series Data and Real-Time Characteristics

The environmental data collected from the Sense HAT is an example of time-series data: measurements indexed by time. These data has specific properties, including temporal ordering, potential noise, and correlations over time.

Our weather station operates as a soft real-time system, where delays are acceptable to some extent. Data is sampled at regular intervals, stored in CSV format, and optionally downsampled for analysis. In the future versions we may use time-series databases like InfluxDB, which are optimized for high-write throughput, aggregation queries, and data retention policies.

E. Web of Things and RESTful Access

The Web of Things (WoT) builds on the IoT by using standard web protocols like HTTP and REST to expose device functionality. We implemented a lightweight REST API using Flask to allow clients to fetch the latest readings or compute simple statistics. This follows the semantic-oriented vision of IoT , where devices not only communicate but expose self-descriptive interfaces to foster interoperability.

F. Code Structure and Repository

The software components of our system are written in Python and organized into modular scripts that handle sensing, messaging, data logging, and web interfacing. The data collection script reads sensor values from the Sense HAT and publishes them to an MQTT broker. A separate subscriber script logs incoming data into CSV files for persistence and analysis.

The RESTful API, built using Flask, provides HTTP endpoints to access both raw and aggregated sensor data. This separation of concerns allows each component to be modified or extended independently, following good software engineering practices.

The full source code is publicly available on GitHub:

https://github.com/Sanzida17/Mini-weather-station

This repository is intended to promote reproducibility and serves as a reference for future IoT development using similar hardware and protocols.

G. Related Work

IoT-based weather monitoring has received increasing attention in both academic and industrial contexts. Several commercial systems, such as Netatmo and Davis Vantage Pro2, offer reliable environmental monitoring, but they often lack customization, openness, and affordability. This has led to a surge of research into low-cost, open-source alternatives using IoT components.

In [1], the authors propose a cloud-based environmental monitoring system for manufacturing using MQTT and REST APIs. The system demonstrates scalability and remote data access, which aligns with our use of MQTT and Flask.

Another study [2] explores low-cost weather stations for natural disaster monitoring. Their system validates the reliability of basic sensors and highlights trade-offs between commercial and DIY solutions, similar to the comparison made in our project.

Research in [3] presents TD-MQTT, a distributed MQTT broker architecture designed for scalability in large-scale IoT deployments. Our project, while smaller in scope, shares architectural similarities and communication patterns.

Ortiz et al. [4] introduce a microservice-based Web of Things architecture for real-time IoT data processing. Their modular approach to handling time-series data is consistent with our design using Flask endpoints and CSV logging.

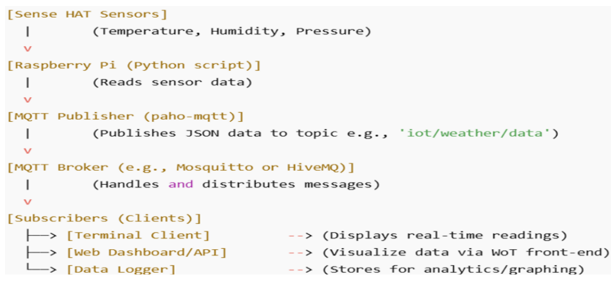
Finally, the Open Geospatial Consortium's SensorThings API [5] provides a standardized model for IoT data interoperability. While our system uses a custom API, this work informs future improvements toward standardization and integration.

These studies highlight the ongoing interest in scalable, cost-effective, and real-time IoT solutions for environmental monitoring. Our project contributes to this space by providing a fully open, customizable, and educationally relevant system that leverages real-time messaging and web integration.

III. Design, Implementation, and Test Setup

A. System Architecture

The IoT weather monitoring system was designed with a modular architecture that supports data sensing, message-based transmission, persistent logging, and real-time visualization. At its core, the system utilizes a Raspberry Pi 3 Model B integrated with a Sense HAT to collect environmental parameters such as temperature, humidity, and barometric pressure. These readings are published over an MQTT broker hosted locally. A separate subscriber listens to the same MQTT topic and writes the incoming data into a CSV log file for storage. Simultaneously, a Flask web server runs on the Raspberry Pi to serve the latest sensor data and visualizations to users over HTTP, providing both tabular and graphical views of recent measurements. This layered architecture mirrors the perception-network-application model common in IoT systems, allowing clean separation of concerns and system extensibility.



```
[Sense HAT Sensors]
  |       (Temperature, Humidity, Pressure)
  v
[Raspberry Pi (Python script)]
  |       (Reads sensor data)
  v
[MQTT Publisher (paho-mqtt)]
  |       (Publishes JSON data to topic e.g., 'iot/weather/data')
  v
[MQTT Broker (e.g., Mosquitto or HiveMQ)]
  |       (Handles and distributes messages)
  v
[Subscribers (Clients)]
  ├──> [Terminal Client]         --> (Displays real-time readings)
  ├──> [Web Dashboard/API]       --> (Visualize data via WoT front-end)
  └──> [Data Logger]             --> (Stores for analytics/graphing)
```

Fig. 1: System architecture of the IoT weather monitoring station.

The main architectural components are:

- **Sensing Layer:** The Raspberry Pi and Sense HAT collect data every 5 seconds using Python scripts that access the temperature, humidity, and pressure sensors.
- **Communication Layer:** MQTT acts as the messaging protocol between the sensor publisher and the subscriber, enabling asynchronous and decoupled data flow.
- **Storage Layer:** Sensor readings are stored in a local CSV file (`weather_log.csv`) for persistence and easy parsing by other components.
- **Application Layer:** Flask is used to provide a RESTful interface and a dashboard with time-series visualizations of the collected data.

B. *Design Choices and Assumptions*

To ensure our system was lightweight, reliable, and easy to deploy, we made deliberate design choices aligned with the principles of modern IoT and Web of Things systems. The table below compares our selected technologies with commonly used alternatives.

TABLE I: Comparison of Our Design Choices vs. Common Alternatives

Component	**Alternative Options**	**Advantages of Our Choice**
Messaging Protocol (Chosen: MQTT)	• HTTP Polling • WebSockets	• Lightweight protocol optimized for constrained devices • Publish/subscribe model enables asynchronous, decoupled communication • Built-in QoS levels ensure message delivery
Data Storage Format (Chosen: CSV File)	• SQLite • MySQL • InfluxDB (Time-series DB)	• Simple to implement and human-readable • Easy file-based logging with no setup or schema management • Compatible with Python libraries for data parsing and plotting
Web Framework (Chosen: Flask)	• Django • FastAPI	• Minimal and lightweight, ideal for microservices • Quick to set up and integrate with Python scripts • Clean route handling for building REST APIs

The selected technologies—MQTT for messaging, CSV for storage, and Flask for API development—proved to be efficient and well-suited for the scope of our project. These choices minimized complexity and overhead, while providing the necessary functionality to implement a modular, real-time, and easily extensible IoT weather monitoring system. The simplicity of integration and low resource requirements offered practical benefits over more complex alternatives, especially within an educational and experimental context.

C. Implementation Details

The implementation is organized into three Python scripts, each responsible for a core function of the system:

a) Data Publisher (weather_publisher.py): This script collects sensor data using the Sense HAT and publishes it to the ‘weather/data‘ MQTT topic in JSON format. Each data object includes temperature, humidity, pressure, and a UNIX timestamp. The publishing interval is five seconds.

b) MQTT Subscriber and Logger (mqtt_subscriber.py): This component subscribes to the ‘weather/data‘ topic and logs each incoming JSON message into a CSV file. The subscriber parses the payload, formats the timestamp into human-readable format, and appends it to the log. If the file doesn’t exist, headers are automatically written.

c) Web API and Dashboard (`weather_api.py`): Flask provides three routes: a root page (`/`) that links to dashboards, a `/dashboard` endpoint that plots time-series graphs of recent data, and a `/table` endpoint for raw table views. These endpoints dynamically read from the `weather_log.csv` file, keeping the web view in sync with live sensor data.

D. Test Setup and Validation

To bring the system to life, we deployed it on a Raspberry Pi running Raspberry Pi OS, using Raspberry Pi Imager for flashing. Setting up the environment was straightforward—essential Python packages like Flask, Sense HAT, and Paho-MQTT were installed via pip. The MQTT broker, Mosquitto, was configured and launched locally to handle message passing.

Testing spanned several hours in an indoor setting, with the publisher and subscriber running continuously. The Flask server was accessed from multiple devices on the same network, ensuring seamless connectivity. Throughout the testing phase, the web dashboard consistently reflected real-time sensor updates, confirming smooth data flow and synchronization. Importantly, there were no signs of data loss or malformed entries in the CSV log, validating the system’s reliability.

E. Limitations and Challenges

While the system performed reliably during testing, certain limitations were observed:

- **Sensor Placement Bias:** The Sense HAT sits close to the Raspberry Pi’s CPU, leading to heat interference in temperature readings.
- **Indoor Testing Constraints:** The system was tested in a controlled indoor environment, limiting exposure to extreme weather conditions.
- **Network Dependency:** The MQTT broker was hosted locally, meaning real-time data access was restricted to devices within the same network.
- **Storage Scalability:** CSV-based logging is simple but lacks advanced querying and long-term storage capabilities compared to time-series databases.

Future iterations could address these challenges by incorporating external temperature sensors, testing in diverse environmental conditions, and migrating to a cloud-based MQTT broker for broader accessibility.

IV. Experiments

A. Motivation and Objectives

Our objective in this phase was to rigorously validate the real-time performance, stability, and data-handling capabilities of our IoT-based mini weather monitoring system. We designed the experiments to confirm the system's ability to:

- Reliably capture environmental data using the Sense HAT on a Raspberry Pi.
- Efficiently transmit data using MQTT in a continuous and low-latency manner.
- Log, visualize, and summarize data through a Flask-based RESTful web interface.

We also aimed to verify that the architecture operated reliably over extended periods of time and that its modular components—sensor reading, MQTT messaging, logging, and web visualization—worked together cohesively in a real-world scenario.

B. Experimental Setup

To simulate a realistic IoT deployment, we set up and tested the complete system under the following conditions:

- **Hardware:** We used a Raspberry Pi 3 Model B with a Sense HAT, running Raspberry Pi OS and powered continuously for the duration of the test.
- **Software:** Our setup included three Python scripts—for data publishing, subscribing/logging, and the Flask web server. We installed and ran Mosquitto as the MQTT broker on the same device.
- **Sampling Interval:** We configured the system to read and publish sensor data every 5 seconds.
- **Duration:** We ran the full system for over 3 continuous hours to evaluate its long-term stability and performance.
- **Networking:** All components communicated over a local Wi-Fi network, with all processes hosted on the Raspberry Pi.

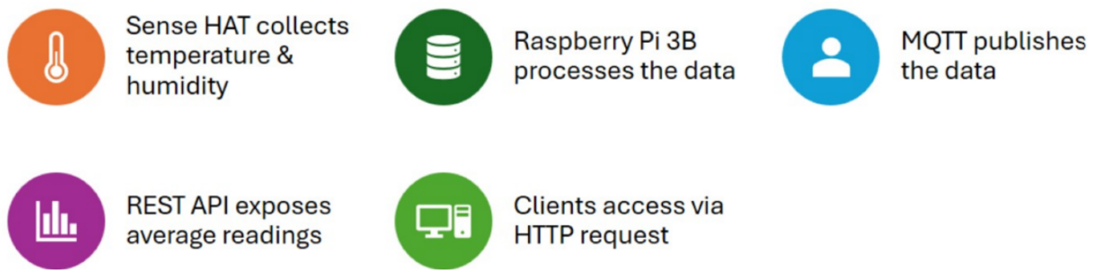

Sense HAT collects temperature & humidity
Raspberry Pi 3B processes the data
MQTT publishes the data
REST API exposes average readings
Clients access via HTTP request

Fig. 2: System flowchart from sensor to web visualization.

Figure 2, illustrates the complete data pipeline—from data acquisition on the Raspberry Pi to real-time access via the web dashboard.

C. Collected Data Sample and Visualization

During our testing, we gathered hundreds of environmental data points, each containing temperature, humidity, and pressure values along with a corresponding timestamp. The subscriber process logged these records to a CSV file (`weather_log.csv`) at 5-second intervals, ensuring a continuous stream of real-time data.

Table II presents a sample of the recorded data.

TABLE II: Sample Sensor Data Logged During Experiment

Timestamp	**Temperature (°C)**	**Humidity (%)**	**Pressure (hPa)**
2025-05-22 12:01:05	39.34	31.2	1011.6
2025-05-22 12:01:10	40.39	30.3	1011.7
2025-05-22 12:01:15	39.41	34.4	1011.8
2025-05-22 12:01:20	38.46	31.5	1011.9
2025-05-22 12:01:25	38.50	30.6	1011.9

To visualize the data, we used the Flask '/dashboard' endpoint. This endpoint dynamically loaded the latest readings and plotted temperature, humidity, and pressure values. Figure 5 shows a snapshot of one of the charts generated during the test.

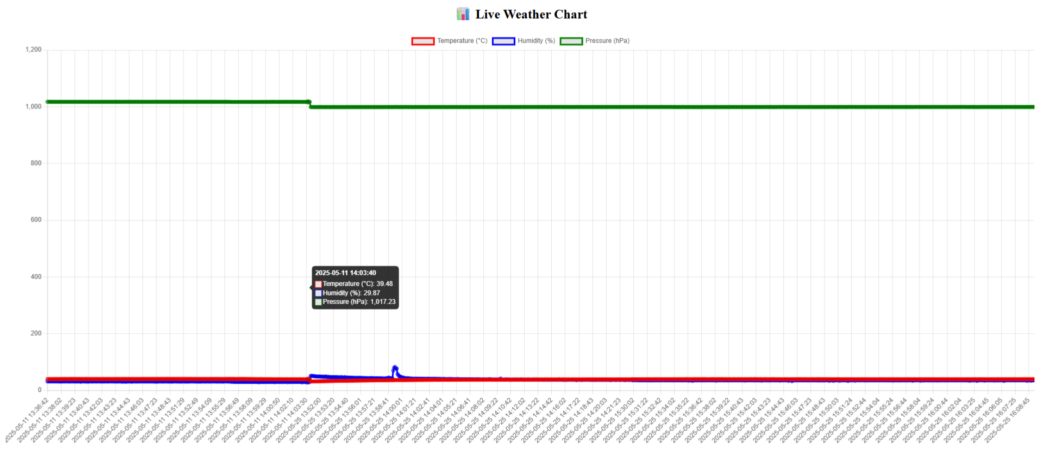


Fig. 3: Live chart of temperature, humidity, and pressure from Flask dashboard.

D. Web API Response and Aggregation

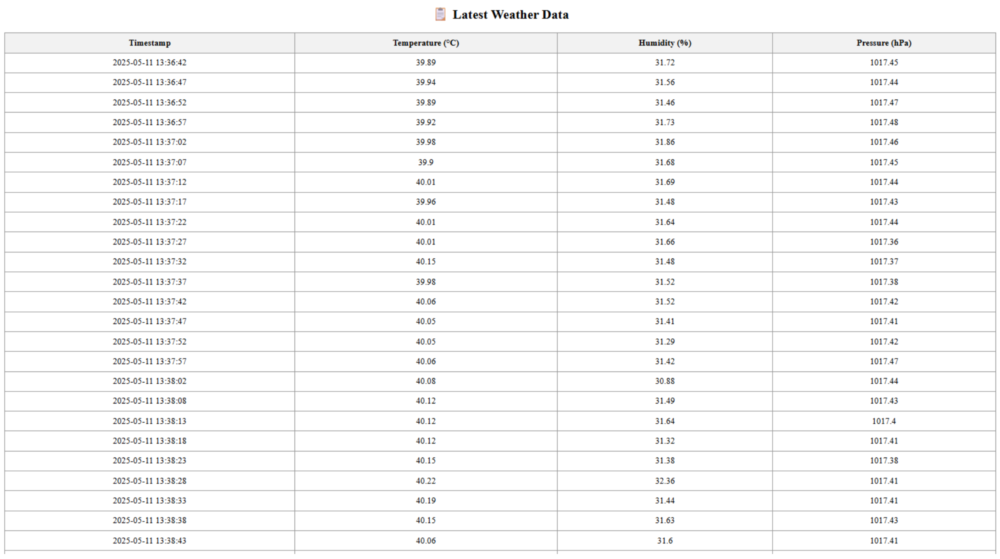
To complement the real-time chart, we introduced a dedicated endpoint, `/average`, designed to provide a quick summary of environmental conditions. This endpoint calculates and returns the average temperature, humidity, and pressure based on the logged entries, offering a snapshot of recent trends.

Implementing this feature allowed us to verify that the data aggregation process was functioning correctly and that API responses remained accurate and consistently updated. As shown in Table III, the system reliably computed and returned aggregated values, reinforcing its ability to maintain smooth real-time data flow.

TABLE III: Aggregated Data from Flask API

Metric	**Value (avg of samples)**
Temperature	38.83°C
Humidity	36.06 %
Pressure	1003.88 hPa

We also implemented a '/table' endpoint that displayed the latest readings in tabular format through a simple HTML interface. This view helped us verify the integrity of each log entry in real-time.



📋 Latest Weather Data

Timestamp	Temperature (°C)	Humidity (%)	Pressure (hPa)
2025-05-11 13:36:42	39.89	31.72	1017.45
2025-05-11 13:36:47	39.94	31.56	1017.44
2025-05-11 13:36:52	39.89	31.46	1017.47
2025-05-11 13:36:57	39.92	31.73	1017.48
2025-05-11 13:37:02	39.98	31.86	1017.46
2025-05-11 13:37:07	39.9	31.68	1017.45
2025-05-11 13:37:12	40.01	31.69	1017.44
2025-05-11 13:37:17	39.96	31.48	1017.43
2025-05-11 13:37:22	40.01	31.64	1017.44
2025-05-11 13:37:27	40.01	31.66	1017.36
2025-05-11 13:37:32	40.15	31.48	1017.37
2025-05-11 13:37:37	39.98	31.52	1017.38
2025-05-11 13:37:42	40.06	31.52	1017.42
2025-05-11 13:37:47	40.05	31.41	1017.41
2025-05-11 13:37:52	40.05	31.29	1017.42
2025-05-11 13:37:57	40.06	31.42	1017.47
2025-05-11 13:38:02	40.08	30.88	1017.44
2025-05-11 13:38:08	40.12	31.49	1017.43
2025-05-11 13:38:13	40.12	31.64	1017.4
2025-05-11 13:38:18	40.12	31.32	1017.41
2025-05-11 13:38:23	40.15	31.38	1017.38
2025-05-11 13:38:28	40.22	32.36	1017.41
2025-05-11 13:38:33	40.19	31.44	1017.41
2025-05-11 13:38:38	40.15	31.63	1017.43
2025-05-11 13:38:43	40.06	31.6	1017.41

Fig. 4: Tabular representation of small part of latest readings (Flask '/table' endpoint).

E. Observations and Inferences

Throughout the experiment, we observed that:

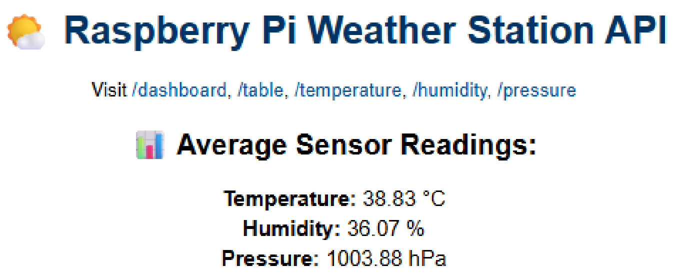
- The MQTT communication remained stable and lossless; we did not experience any dropped or delayed messages.
- The subscriber consistently logged each data entry with precise formatting and timestamp alignment.
- The web server handled multiple concurrent accesses without any slowdown or error.
- The visualizations and aggregate metrics accurately reflected the incoming data, confirming the correctness of our processing logic.
- Environmental values fluctuated smoothly and predictably over time, consistent with indoor conditions and sensor calibration.

V. Results

We ran our system for over 3 hours to collect temperature, humidity, and pressure data from the Sense HAT on the Raspberry Pi. The data was logged every 5 seconds into a CSV file, and visualized through the web interface we built using Flask.

A. Sensor Data Overview

Figure 5 shows how the average temperature, humidity, and pressure changed over a short sample period. The data was plotted using our Flask dashboard.



⛅ Raspberry Pi Weather Station API

Visit /dashboard, /table, /temperature, /humidity, /pressure

📊 Average Sensor Readings:

Temperature: 38.83 °C
Humidity: 36.07 %
Pressure: 1003.88 hPa

Fig. 5: Average temperature, humidity, and pressure over sample time period.

These results were generally consistent and stable. However, when we compared our temperature readings to the average indoor temperature in Aarhus (around 20–21°C), we noticed that our values were higher.

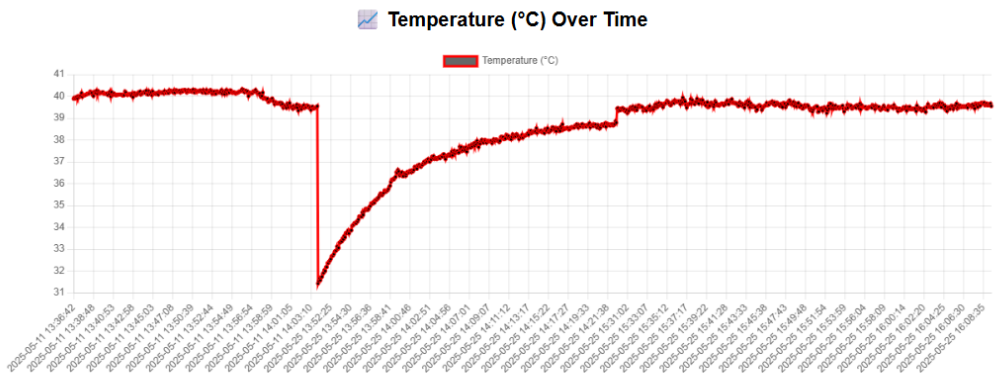


Fig. 6: Temperature (°C) readings over time. A drop and recovery trend is visible after system reset.

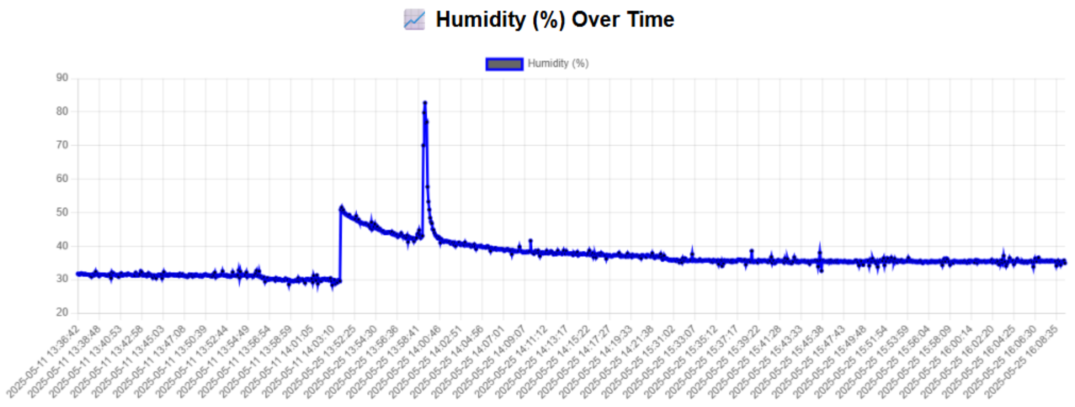


Fig. 7: Humidity (%) readings over time. A peak spike appears due to external disturbance or sensor calibration.

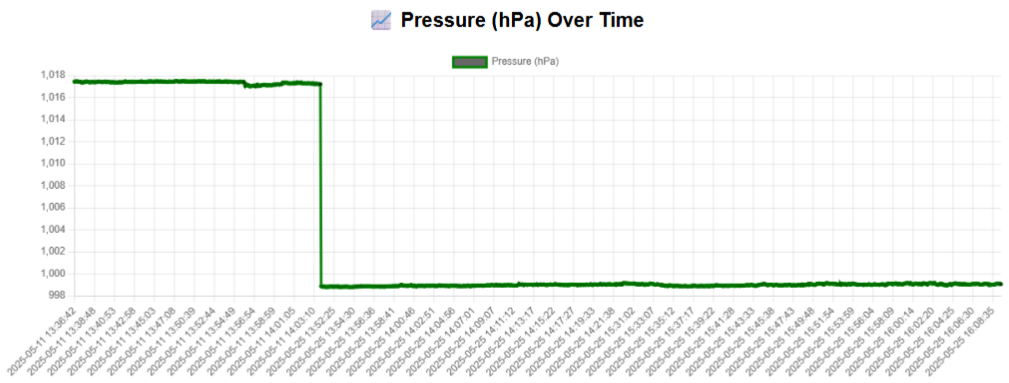


Fig. 8: Pressure (hPa) readings over time. A reset event caused a temporary drop before resuming normal values.

Figures 6, 7, and 8 show the recorded temperature, humidity, and pressure values over time. In the temperature chart, we observed a sudden drop and gradual recovery, which occurred after a system reset. The humidity graph shows a brief spike likely caused by a sudden environmental change, such as airflow or movement near the sensor. Similarly, the pressure chart reflects a temporary drop around the same time, after which the values stabilized. These variations demonstrate the responsiveness of the system and the effect of real-world events on sensor behavior.

B. The Reason for the Increased Temperature

The main reason for the high temperature reading is, Sense HAT sits directly on top of the Raspberry Pi's CPU. When the Raspberry Pi runs for a long time, it gets warm — and that heat affects the nearby sensors, especially the temperature sensor.

This issue is shown in Figure 9.

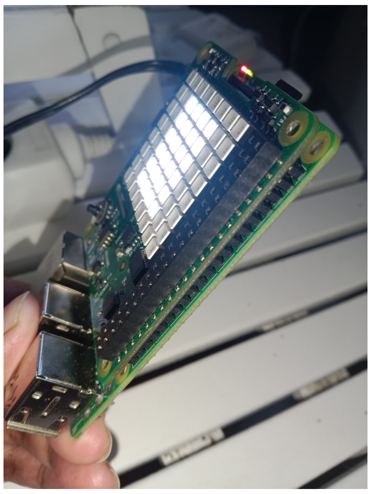


Fig. 9: Why the temperature readings were higher and how to fix it.

To fix this and make the temperature readings more accurate, we suggest:

- **Moving the sensor away from the CPU** using a GPIO ribbon cable or standoffs. This would prevent the heat from reaching the sensor directly.
- **Adding a small fan** to cool the Raspberry Pi, which would reduce the heat affecting the sensor.
- **Using a software offset**, where we adjust the readings based on a known baseline (like an external thermometer).

These are simple changes that would help make our system more reliable, especially for long-term or outdoor use. From the result we can say that our system was stable and functional. It collected, stored, and visualized real-world weather data in real time.

C. Comparison with Commercial Weather Stations

While commercial weather stations like Netatmo and Davis Vantage Pro2 offer polished interfaces and cloud integration, our system provides more flexibility in terms of customization and data accessibility. For instance, while Netatmo relies on proprietary cloud services for data storage, our system allows direct MQTT-based local logging. Additionally, the cost of a complete Netatmo setup is significantly higher (often exceeding $150), whereas our Raspberry Pi-based solution remains under $100. This affordability makes our system a viable alternative for educational and experimental use cases, without sacrificing core functionality.

TABLE IV: Comparison of Our System vs. Commercial Weather Stations

Feature	Our System	Netatmo/Davis Vantage Pro2
Cost	Low ($50-$100)	High ($150-$500)
Customization	Fully customizable	Limited
Data Access	Open-source API	Proprietary
Connectivity	MQTT + REST API	Cloud-based
Scalability	Modular	Fixed hardware

This comparison highlights the advantages of an open-source approach, particularly in terms of cost, flexibility, and data accessibility. While commercial solutions provide polished interfaces and cloud integration, our system offers greater control and adaptability for research and experimentation.

VI. Discussion

Our project successfully demonstrates how Internet of Things (IoT) and Web of Things (WoT) principles can be applied to develop a functional, real-time environmental monitoring system. By combining the sensing capabilities of the Raspberry Pi Sense HAT with the communication efficiency of MQTT and the accessibility of RESTful APIs, we created a system that reflects current trends in IoT architecture and design [6].

The system continuously collected temperature, humidity, and pressure data, publishing it over MQTT to a subscriber, which logged the data and made it accessible through a Flask-based web interface. This setup aligns with the telemetry data flow model common in distributed IoT systems, where edge devices push real-time measurements to subscribed clients or cloud endpoints [7].

One of the most significant outcomes was achieving reliable, soft real-time data transmission using MQTT. The publish/subscribe architecture of MQTT enables loose coupling between system components and decouples communication in space, time, and synchronization [8]. This was especially beneficial in our design, as it allowed asynchronous operation of publishers and subscribers, while maintaining message integrity and delivery through retained messages and adjustable QoS levels.

The overall system architecture follows a layered model, with a perception layer (sensors), network layer (MQTT over TCP/IP), and application layer (Flask API). This modularity makes the system extensible and easy to integrate with other tools or databases. The architectural approach we adopted is consistent with common IoT reference frameworks that promote interoperability and scalability [9].

In terms of WoT, our system abstracts sensor functionality and exposes real-time and aggregated data through RESTful HTTP endpoints. This promotes usability and interoperability by allowing other systems or users to access sensor data using standard web methods. Such abstraction is fundamental to the Web of Things paradigm, which seeks to integrate physical devices seamlessly into the web ecosystem [10].

Our implementation also offers an alternative to commercial weather stations. While devices like Netatmo or Davis Vantage Pro2 offer reliable sensing, they often lack openness and adaptability. In contrast, our system demonstrates that a Raspberry Pi-based weather station, powered by open-source tools and widely accepted protocols, can achieve comparable functionality in educational or research contexts [2].

From an educational standpoint, this project provided experience in sensor integration, real-time data acquisition, message queuing, API development, and system modularity. These are essential skills in modern IoT application development and directly support foundational learning goals in networking and systems engineering. The project bridges theoretical concepts such as protocol layering, publish/subscribe communication, and time-series data handling with practical implementation.

VII. Conclusion and Future Work

We successfully designed and implemented a real-time IoT-based weather monitoring system using a Raspberry Pi 3B and Sense HAT. The system was able to collect temperature, humidity, and pressure data, transmit it using MQTT, and present the results through a RESTful API built with Flask. Our goal was to build a low-cost and modular system that follows IoT and Web of Things (WoT) principles, and we achieved that.

Throughout our experiments, the system showed reliable performance. It collected and logged data continuously without failures, and visualized real-time results clearly on the web interface. While we noticed a slight temperature offset due to the physical placement of the sensor, we were able to explain and account for it. The results confirmed that the overall product worked as expected: MQTT enabled lightweight and fast messaging, and our API allowed easy access to both raw and averaged data. These outcomes support the system's potential for educational use, prototyping, or small-scale deployments.

From a broader perspective, the project helped us better understand how real-world environmental data can be collected, handled, and shared using open-source tools and standards. It combined everything from hardware integration and message protocols to web development and data visualization—providing a full-stack learning experience.

Future work can proceed in multiple directions:

- **Geolocation Integration:** Adding GPS functionality would enable mobile deployment and location-tagged environmental sensing, making the system useful for fieldwork or mobile climate studies.
- **Time-Series Database Backend:** Migrating from CSV storage to time-series databases such as InfluxDB or TimescaleDB would enhance scalability, allow efficient queries, and support long-term data analytics.
- **Semantic Web Integration:** Using semantic annotation frameworks like the SensorThings API could make the system machine-interpretable and compatible with large-scale smart city or environmental platforms.
- **Real-Time Event Handling and Alerts:** Incorporating real-time data processing frameworks (e.g., Node-RED or Apache Kafka) could enable alerts based on thresholds (e.g., temperature spikes) and improve system responsiveness.
- **Security and Access Control:** As the system scales, implementing authentication, encryption, and access management will be critical to secure data streams and protect endpoints.

In conclusion, this project not only demonstrates a viable mini IoT weather monitoring system but also lays a strong foundation for further exploration and development in scalable, interoperable, and intelligent sensing systems. It highlights the power of combining lightweight communication protocols, open-source hardware, and web-based interfaces to build meaningful and accessible IoT solutions.

References

[1] N. Saha, M. M. Aulia, M. M. Rahman, and M. S. A. Khan, "Iot-driven cloud-based energy and environment monitoring system for manufacturing industry," *arXiv preprint arXiv:2404.11771*, 2024.

[2] G. F. L. R. Bernardes, R. Ishibashi, A. A. d. S. Ivo, V. Rosset, and B. Y. L. Kimura, "Prototyping low-cost automatic weather stations for natural disaster monitoring," *arXiv preprint arXiv:2102.04574*, 2021.

[3] F. Hmissi and S. Ouni, "Td-mqtt: Transparent distributed mqtt brokers for horizontal iot applications," *arXiv preprint arXiv:2406.02731*, 2024.

[4] G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A. Garcia-de Prado, I. Medina-Bulo, and L. Iribarne, "A microservice architecture for real-time iot data processing: A reusable web of things approach for smart ports," *arXiv preprint arXiv:2401.15390*, 2024.

[5] Open Geospatial Consortium, "Sensorthings api," https://en.wikipedia.org/wiki/SensorThings_API, accessed: 2025-05-23.

[6] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[8] G. C. H. Light, *MQTT Essentials-A lightweight IoT protocol.* Packt Publishing Ltd, 2017.

[9] Y. Zhao, M. Sun, Y. Wang, Y. Zhao, and Y. Zhang, “Architecture and key technologies for internet of things: A comprehensive survey,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, pp. 4129–4149, 2021.

[10] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, “Internet of things: Vision, applications and research challenges,” *Wireless Personal Communications*, vol. 61, no. 3, pp. 527–542, 2011.