Mongo Secure Climate Logger

Report Submitted to
IoT Academy
IIT Guwahati
Assam Skill Development Mission

For the fulfilment of the requirements of the Award of Certification for the program
Training on Industrial IoT and Edge AI Analyst

By

Ankan Debnath
Ravijyoti Motak
Dildar Matin Piar

Under the Supervision of
IoT Academy

(Institute of technology in Noida, Uttar Pradesh)

DECLARATION

We declare that

I. The work contained in this project is original and has been done by us under the guidance of our supervisor.
II. The work has not been submitted to any other Institute for any degree or diploma.
III. We have followed the guidelines provide by the Institute in preparing the report.
IV. We have conformed to the norms and guidelines given in the Ethical code of Conduct of the Institute.

ANKAN DEBNATH
RAVIJYOTI MOTAK
DILDAR MATIN PIAR

CERTIFICATE

*This is certified that the Report entitled, "**Mongo Secure Climate Logger**" submitted by **ANKAN DEBNATH**, **RAVIJYOTI MOTAK** and **DILDAR MATIN PIAR** to the IoT Academy, is a record of bonafide Project work carried out by us under the supervision, guidance and worthy of consideration for the award for certification on Training on Industrial IoT and Edge AI analyst.*

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to all those who have contributed to the successful completion of this project. Their guidance, support, and encouragement have been invaluable throughout this journey.

I am deeply thankful to **Mr. Sanjeev Barman** for his constant guidance, insightful suggestions, and unwavering support. His expertise and mentorship have played a pivotal role in shaping this project and helping me overcome challenges.

I would also like to extend my sincere thanks to **Mrs. Gayatri Banik** for her invaluable assistance, constructive feedback, and encouragement. Her timely advice and support have greatly contributed to the successful completion of this work.

A special thanks to **Mr. Vicky Mesh**, Research Scholar, for his invaluable contribution in providing the idea of the online dashboard, which displays the data effectively. His innovative suggestions and technical insights have greatly enhanced the functionality and presentation of this project.

Lastly, I am grateful to my friends, family, and colleagues for their encouragement and support throughout the project. This accomplishment would not have been possible without their help.

ABSTRACT

The rapid expansion of Internet of Things (IoT) technologies underscores the need for efficient and reliable sensor data acquisition systems. This project introduces an integrated environmental monitoring solution using an ESP8266 microcontroller and a DHT11 sensor, developed with the Arduino IDE. The system captures real-time temperature, humidity, and heat index measurements, formatting them into JSON for transmission via Wi-Fi to a RESTful API server. The data is then stored in a MongoDB Atlas database, managed using MongoDB Compass, ensuring persistent and scalable record-keeping. The system's cost-effectiveness and modular design demonstrate robust hardware-software integration, maintaining high data accuracy and transmission reliability. Extensive testing confirms timely sensor data delivery, suitable for smart home monitoring to industrial automation. The collected data is visualized using ThingSpeak, providing real-time analytics and enhancing usability.

To bolster security, the project incorporates pfSense, an open-source firewall and router software. pfSense offers advanced features like stateful packet inspection, VPN support, traffic shaping, and high availability, protecting the ESP8266 sensor, REST API server, and MongoDB database from unauthorized access and cyber threats. This integration ensures the system remains secure and reliable, addressing current sensor networking challenges and laying a foundation for future enhancements like real-time analytics and automated alerts.

Keywords: DHT11 sensor, ESP8266, Arduino IDE, REST API, MongoDB, MongoDB Compass, environmental monitoring, IoT, heat-index calculation, ThingSpeak, pfSense, network security, firewall, VPN.

1. DHT Sensor

The DHT11 is a low-cost digital sensor used to measure temperature and humidity. It is commonly used in Internet of Things (IoT) applications where such environmental data needs to be monitored or transmitted. The DHT11 can be connected to microcontrollers (like Arduino or Raspberry Pi) to collect data and perform actions based on environmental conditions.

1.1 Key Features of the DHT11 Sensor:

1. **Temperature Range**: 0°C to 50°C (accuracy ±2°C)
2. **Humidity Range**: 20% to 90% RH (accuracy ±5% RH)
3. **Output**: Digital signal (using a single-wire protocol)
4. **Power Supply**: Typically 3.3V to 5V DC
5. **Communication Protocol**: Single-wire serial interface

1.2 Basic Working Principle:

- The DHT11 has a sensor component inside that detects temperature and humidity. It converts the readings into digital data, which is then transmitted over a single-wire data line to a microcontroller.
- The sensor has a built-in microcontroller that handles the process of converting the analog signals from the humidity and temperature sensing elements into digital values.

1.3 How DHT11 Works:

1. **Initial Setup**: The DHT11 is powered by a DC voltage source (3.3V or 5V).
2. **Communication**: A microcontroller sends a request signal (a start pulse) to the DHT11.
3. **Response**: The DHT11 then sends back a series of bits representing the temperature and humidity readings.
4. **Data Processing**: The microcontroller processes the data and converts it into human-readable values (temperature in °C, humidity in %).

1.4 Pin Configuration:

- **Pin 1 (VCC)**: Power supply (3.3V or 5V)
- **Pin 2 (Data)**: Data pin (used to send and receive information)
- **Pin 3 (NC)**: No connection (not used)
- **Pin 4 (GND)**: Ground

Here’s a simple figure illustrating the DHT11 sensor and its connections:

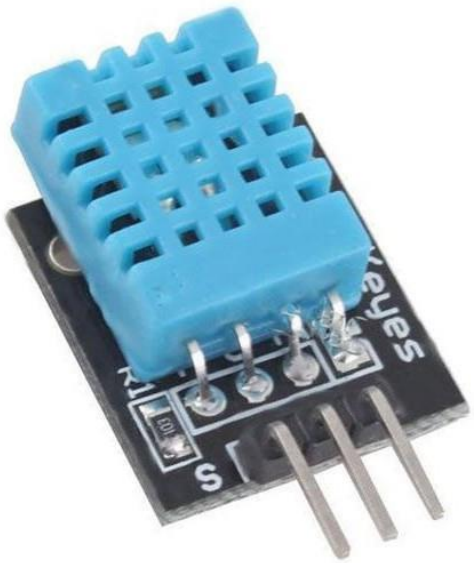

Fig:-DHT11 IoT Sensor

1.5 Application in IoT:

- **Home Automation**: Monitor indoor climate and adjust HVAC systems.
- **Weather Stations**: Measure temperature and humidity to predict weather.
- **Smart Agriculture**: Monitor soil and ambient conditions for optimal plant growth.

With its simple interface and affordable price, the DHT11 is widely used in many IoT projects to collect environmental data.

2. ESP8266

The **ESP8266** is a low-cost Wi-Fi module used to connect microcontrollers (like Arduino) to the internet or a local network. It can handle both client and server functions, making it highly versatile for IoT (Internet of Things) applications. It has a built-in TCP/IP stack, making it suitable for networking tasks in embedded systems.

2.1 Key Features of ESP8266:

1. **Wi-Fi Connectivity**: Provides wireless access to a network.
2. **Operating Voltage**: Typically 3.3V.
3. **Built-in TCP/IP Stack**: Handles network protocols directly on the chip.
4. **GPIO Pins**: Can be used for digital I/O, PWM, ADC, etc.
5. **Low Power Consumption**: It is designed for low-power IoT applications.
6. **Small Size**: Perfect for embedded systems where space is a concern.
7. **Support for Arduino**: It can be programmed using the Arduino IDE.

2.2 Pin Configuration:

While there are several ESP8266 modules (e.g., ESP-01, ESP-12), they share similar features and pinouts. Here's a basic idea of the pinout for the **ESP8266** module:

1. **VCC**: Power supply (3.3V)
2. **GND**: Ground
3. **CH_PD**: Chip enable (needs to be high for the module to function)
4. **TX/RX**: Serial communication (used for programming and data transfer)
5. **GPIO0**: General-purpose input/output (can be used for various functions)
6. **GPIO2**: Another general-purpose I/O pin
7. **RST**: Reset pin
8. **ADC**: Analog-to-digital converter (optional, but available on some models)

Fig: -ESP8266 Microcontroller

2.3 Basic Working Principle:

The ESP8266 communicates with microcontrollers through UART (serial communication). It can act as a Wi-Fi "shield," allowing the microcontroller to send and receive data over the internet or a local network. The ESP8266 can be programmed to handle tasks like connecting to a Wi-Fi network, sending HTTP requests, controlling devices remotely, etc.

How ESP8266 Works:

1. **Powering up**: The ESP8266 module is powered by 3.3V.
2. **Connecting to Wi-Fi**: The module connects to the Wi-Fi network by specifying the network's SSID (name) and password.
3. **Communication**: The ESP8266 sends and receives data over the internet. For example, it can interact with web APIs or send data to a cloud service.
4. **Programming**: You can upload code to the ESP8266 through the UART interface, allowing you to control its functionality.

Example of ESP8266 Wi-Fi Application:

- **Home Automation**: Use ESP8266 to control lights, fans, or other appliances remotely via a mobile app or web interface.
- **IoT Projects**: Send sensor data (e.g., from temperature or humidity sensors) to cloud services for monitoring or processing.

Example Figure of ESP8266 Module:

Common ESP8266 Modules:

1. **ESP-01**: The simplest version, with minimal GPIO pins. Ideal for basic tasks where minimal interfacing is required.
2. **ESP-12**: A more advanced version with more GPIO pins and better performance. It’s often used in more complex IoT applications.

3. Arduino

3.1 What is Arduino IDE?

- The **Arduino IDE (Integrated Development Environment)** is a software application used to write, compile, and upload code to Arduino boards.
- It is a cross-platform application (Windows, macOS, Linux) and is designed to be user-friendly, especially for beginners.
- The IDE provides a simple interface for writing code, debugging, and interacting with Arduino hardware.

3.2 Key Features of Arduino IDE

- **Code Editor**: A text editor with syntax highlighting for writing Arduino sketches (programs).
- **Serial Monitor**: A tool to communicate with the Arduino board and view data sent from it.
- **Library Manager**: Easily install and manage third-party libraries for additional functionality.
- **Board Manager**: Configure and add support for different Arduino boards and architectures.
- **Built-in Examples**: Pre-written code examples to help users learn and implement common tasks.
- **Compilation and Upload**: Compiles code and uploads it to the connected Arduino board via USB.

3.3 What is an Arduino Sketch?

1. An **Arduino Sketch** is the name given to a program written in the Arduino IDE.
    - It is a file with the `.ino` extension that contains code written in C/C++.
    - Sketches control the behavior of the Arduino board by defining inputs, outputs, and logic.

2. Structure of an Arduino Sketch

Every Arduino sketch has two main functions:

- **setup()**: Runs once when the sketch starts. Used to initialize variables, pin modes, and libraries.
- **loop()**: Runs repeatedly after setup(). Contains the main logic of the program.

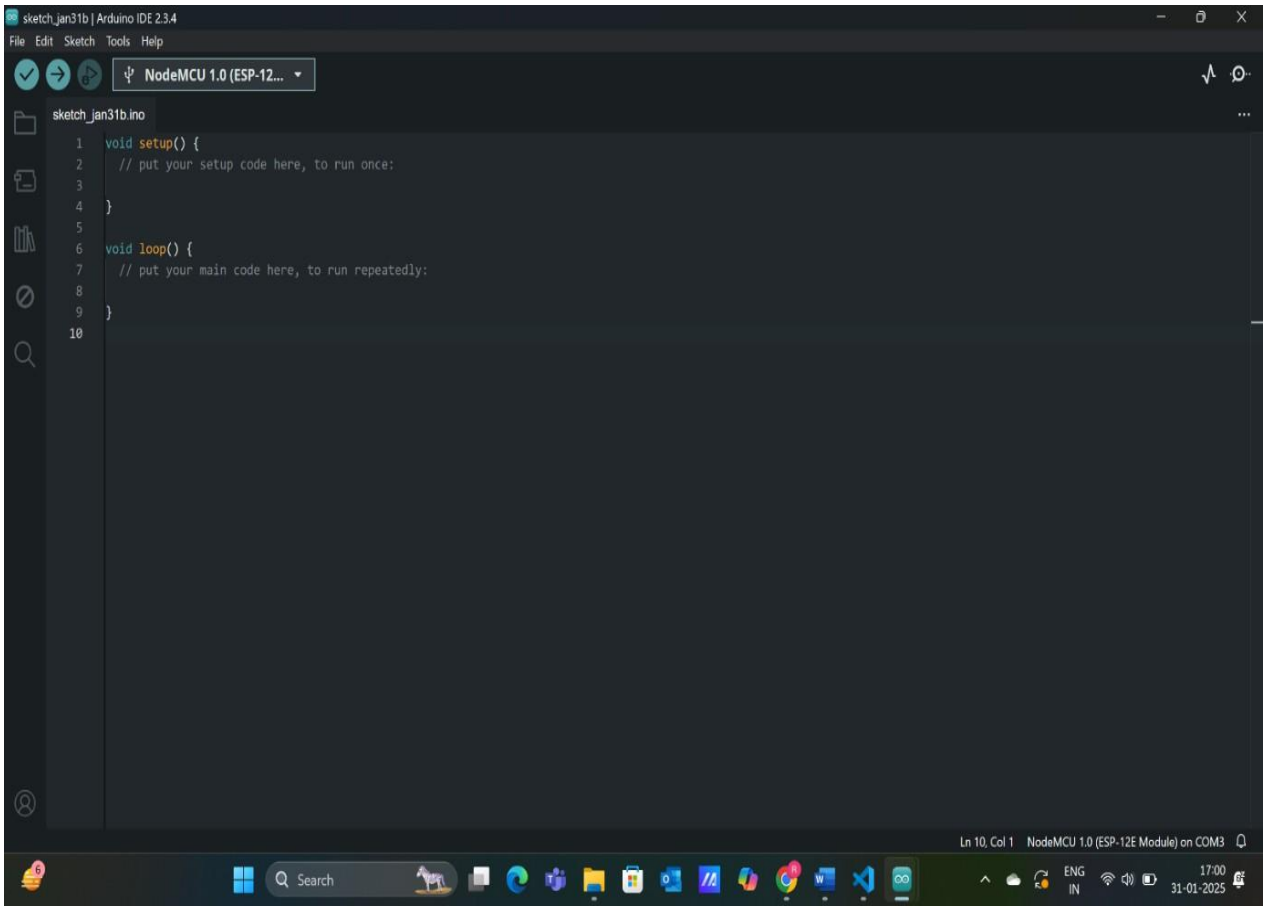

Fig: Arduino IDE Sketch

4. REST API Server

The core of this project is a REST API server that facilitates communication between the ESP8266 microcontroller and a MongoDB database. The server is built using **Node.js, Express.js, and Mongoose**, enabling seamless data transfer and storage.

a. Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It simplifies interactions with MongoDB by providing a schema-based approach to data handling.

i. Allows defining a **schema** to enforce structure on the database.
ii. Provides methods for querying and manipulating MongoDB documents efficiently.
iii. Supports middleware and hooks, making it easy to add validations and logic before saving data.

In this project, we use Mongoose to define a schema for IoT sensor readings, ensuring that data is stored in a structured and manageable format.

---

b. Express.js

Express.js is a lightweight and flexible web application framework for Node.js. It simplifies API development by providing middleware and routing capabilities.

Key reasons for using Express.js in this project:

i. **Lightweight and fast** – Enables handling HTTP requests efficiently.
ii. **Middleware support** – Allows processing request data before reaching the endpoint.
iii. **Simple routing** – Easily defines REST API endpoints for handling sensor data.

Our Express-based server listens for incoming HTTP `POST` requests from the ESP8266 and stores the received sensor data in MongoDB.

---

c. JavaScript in Backend Development

JavaScript is the primary language for this project's backend. Reasons for choosing JavaScript include:

i. **Non-blocking I/O model** (via Node.js), making it ideal for handling multiple incoming sensor data requests simultaneously.

ii. **Vast ecosystem** with frameworks like Express.js and Mongoose that simplify backend development.
iii. **Cross-platform compatibility**, enabling deployment on various cloud platforms and local servers.

---

d. Schema Design and Mongoose Model

The structure of the sensor data is defined using **Mongoose Schema**.

```
const sensorSchema = new mongoose.Schema({
    timestamp: { type: Date, default: Date.now },
    temperature_in_c: Number,
    temperature_in_f: Number,
    humidity: Number,
    heat_index_in_c: Number,
    heat_index_in_f: Number
});
```

This schema ensures that:

1. Each record is **time-stamped** for time-series analysis.
2. **Temperature, humidity, and heat index** values are stored in both Celsius and Fahrenheit.
3. The data format remains **consistent**, preventing incorrect data types from being inserted into the database.

---

5. MongoDB

MongoDB is a **NoSQL (non-relational) database** optimized for handling large-scale, unstructured, and time-series data. In this project, MongoDB stores sensor readings, ensuring efficient retrieval and analysis.

a. Why Use a Non-Relational Database?

Traditional relational databases (e.g., MySQL) store data in structured tables. However, IoT applications deal with **high-frequency, real-time sensor data**, making non-relational databases like MongoDB a better choice because:

ı. **Scalability** – MongoDB scales horizontally, handling large data volumes efficiently.
ıı. **Flexible schema** – No need for predefined tables; new data types can be added dynamically.
ııı. **Faster writes** – MongoDB supports high-speed insertions, essential for continuous IoT data streams.

b. Why Use MongoDB for Time-Series Data?

Time-series data refers to data points collected over time, such as sensor readings. MongoDB is ideal for time-series data because:

1. **Efficient indexing** – Built-in time-based indexes allow fast lookups.
2. **Aggregation framework** – Supports advanced queries for analyzing trends over time.
3. **Data compression** – Reduces storage space by grouping data in BSON format.

---

c. Data Storage Methods

MongoDB stores data in a **JSON-like format (BSON)**, making it easy to work with IoT devices that send data in JSON.

Example of stored sensor data:

```
{
    "_id": "660d4b0b9f93a8b1a5e3f7b5",
    "timestamp": "2024-02-02T10:30:00Z",
    "temperature_in_c": 25.3,
    "temperature_in_f": 77.54,
    "humidity": 60,
    "heat_index_in_c": 26.1,
    "heat_index_in_f": 78.98
}
```

This format ensures each record contains a timestamp for **chronological tracking and analysis**.

---

d. Tools for Time-Series Analysis in MongoDB

MongoDB offers powerful tools for analyzing time-series data, including:

i. **Aggregation Pipeline** – Allows filtering, grouping, and analyzing data over time.
ii. **Time-Series Collections** – Optimized for high-frequency time-based data storage.
iii. **Indexing & Query Optimization** – Ensures efficient data retrieval.

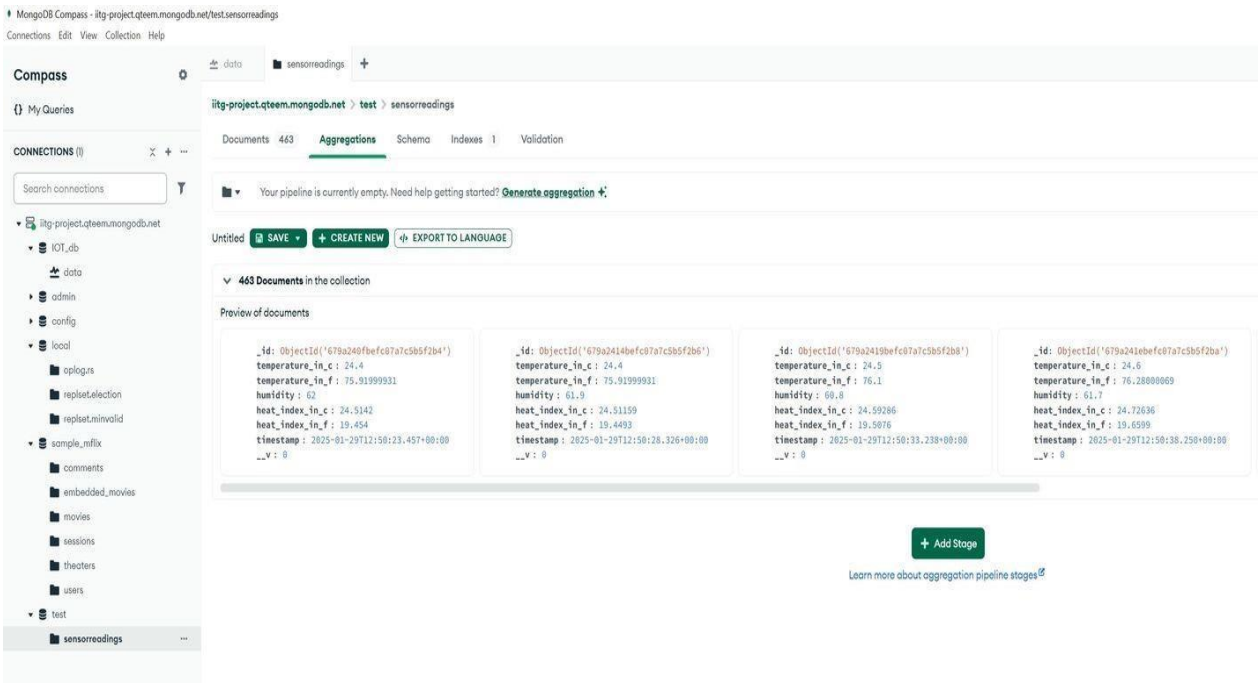

Fig: Stored sensor-readings data in MongoDB database.

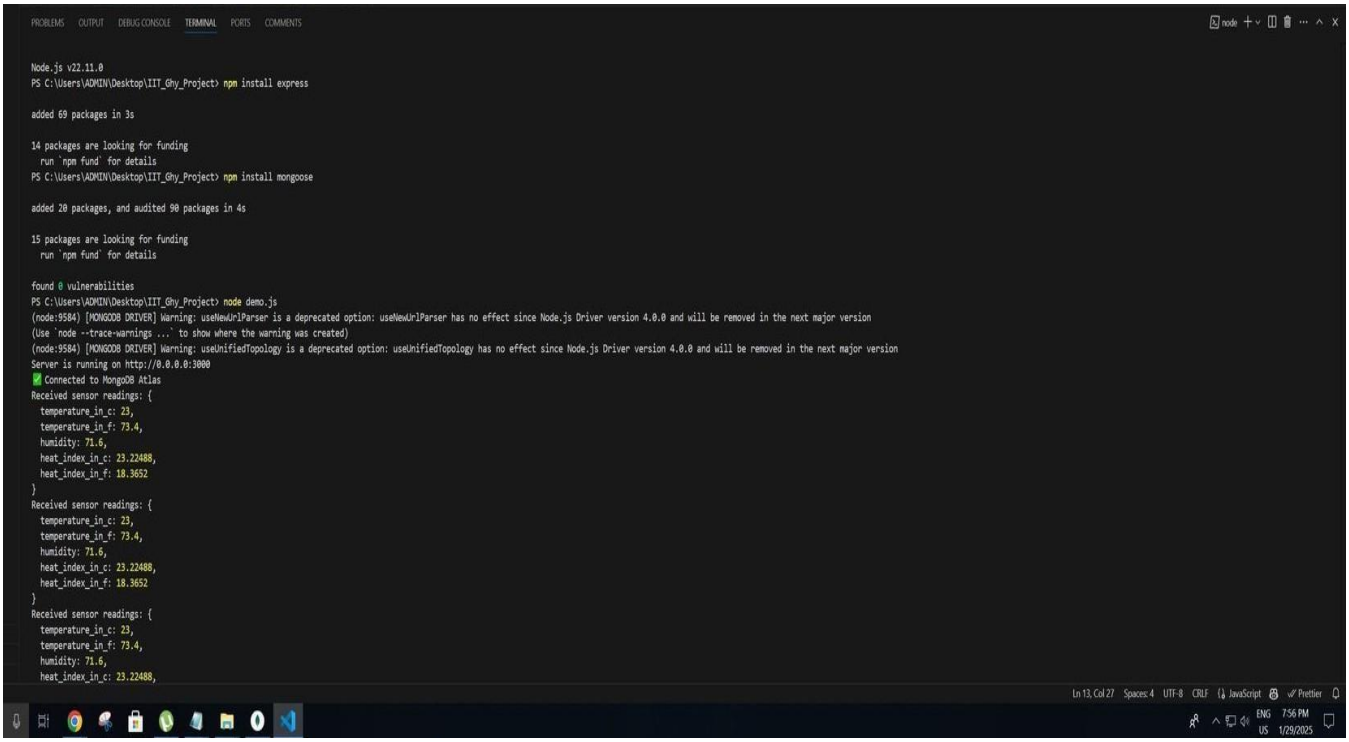

Fig: VSCode output.

6. ThingSpeak

A Python script was used to **upload sensor data** (temperature and humidity) from a JSON file to **ThingSpeak**, a cloud-based IoT analytics platform. The script automates the process of sending sensor readings and allows users to visualize the data in real-time.

a. Data Uploading to ThingSpeak

ThingSpeak requires an **API-key** to authenticate and accept data from external sources. The script uses the API key to send **temperature and humidity** values, which are stored in **Field 1** and **Field 2** of the ThingSpeak channel.

Each data entry in ThingSpeak consists of:

i. **Timestamp** – The time when the data was received.
ii. **Field 1 (Temperature)** – The recorded temperature value.
iii. **Field 2 (Humidity)** – The recorded humidity value.

ThingSpeak automatically **stores, processes, and visualizes** the data in real time.

---

b. Visualization on ThingSpeak

After the script uploads the data, **ThingSpeak automatically generates real-time visualizations**.

1. **Temperature Graph (Field 1)**

- The first graph represents **temperature variations** over time.
- The red line shows **increasing or decreasing trends** in temperature readings.

2. **Humidity Graph (Field 2)**

- The second graph represents **humidity changes** over time.
- The data points indicate fluctuations in **humidity levels**.

These graphs help **monitor sensor data trends**, making it easier to analyze changes in environmental conditions.

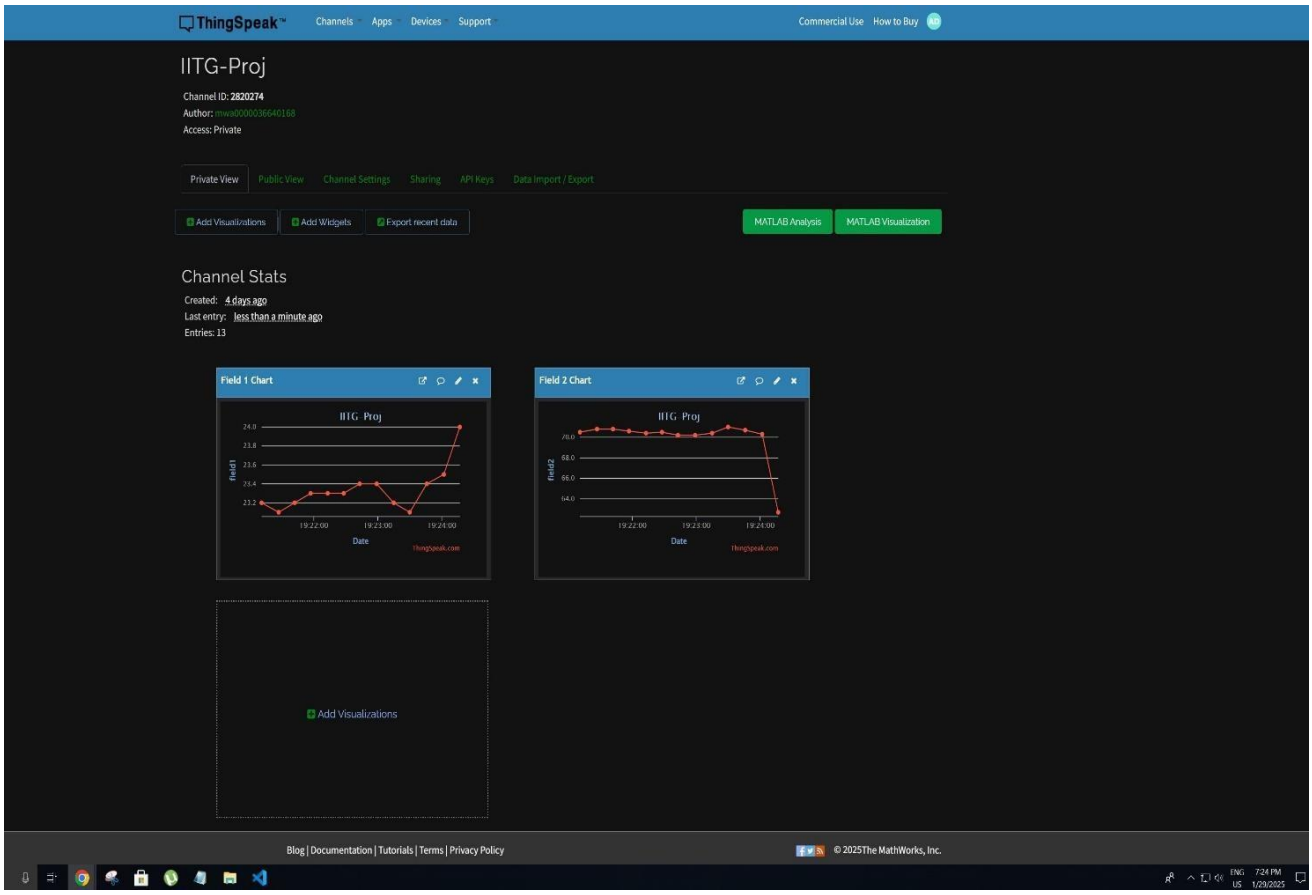

Fig: ThingSpeak (Online) visualization of obtained data.

7. Security Concerns

7.1 Introduction

a. What is pfSense?

i. pfSense is a free, open-source firewall and router software based on FreeBSD.

ii. It provides a wide range of networking features, including firewall, VPN, routing, and traffic shaping.

iii. It is widely used in small to large enterprises, data centers, and home networks.

b. Purpose of the Report

i. To explore pfSense, its features, and its practical applications in network management.

7.2 Key Features of pfSense

c. **Firewall**: Stateful packet inspection, NAT (Network Address Translation), and rule-based traffic filtering.

d. **VPN Support**: OpenVPN, IPsec, and PPTP for secure remote access.

e. **Routing**: Static and dynamic routing (RIP, OSPF, BGP).

f. **Traffic Shaping**: Quality of Service (QoS) to prioritize network traffic.

g. **High Availability**: Failover and load balancing for uninterrupted network services.

h. **Captive Portal**: Authentication for guest networks.

i. **DNS/DHCP Server**: Integrated DNS and DHCP services.

j. **Reporting and Monitoring**: Real-time traffic graphs and logs.

k. **Package System**: Extend functionality with add-ons like Snort (IDS), Squid (proxy), and more.

7.3 Benefits of Using pfSense

- **Cost-Effective**: Free and open-source, reducing licensing costs.
- **Customizability**: Highly configurable to meet specific network requirements.
- **Security**: Regular updates and a strong focus on network security.
- **Scalability**: Suitable for small home networks to large enterprise environments.
- **Community and Support**: Active community and professional support options.

7.4 Use Cases of pfSense

- **Home Networks**: Secure internet access, parental controls, and guest networks.
- **Small Businesses**: Affordable firewall and VPN solution.
- **Enterprise Networks**: High availability, load balancing, and advanced security.
- **ISPs**: Traffic shaping and bandwidth management.
- **Educational Institutions**: Captive portals for Wi-Fi access and content filtering.

7.5 Challenges and Limitations

- **Hardware Dependency**: Performance depends on the hardware used.
- **Learning Curve**: Advanced features may require technical expertise.
- **Updates**: Regular updates are necessary to maintain security.

7.6 How pfSense Enhances Security for Mongo Secure Climate Logger

In our IoT project, **Mongo Secure Climate Logger**, pfSense plays a crucial role in strengthening the security of the system by protecting **sensor data, API communication, and the MongoDB database**. Below are the key ways pfSense enhances security:

1. Firewall Protection for IoT Devices & API Server

- pfSense acts as a **network firewall**, blocking **unauthorized traffic** from accessing the **ESP8266 sensor, REST API server, and MongoDB**.
- **Custom firewall rules** can be set to allow only **whitelisted IPs** to communicate with the server and database.
- Protects against **DDoS attacks** and **malicious requests** trying to exploit vulnerabilities in the API.

2. VPN for Secure Remote Access

- **OpenVPN/IPsec** on pfSense allows **secure remote access** to the IoT system, ensuring that only authenticated users can monitor or modify data.
- Prevents unauthorized external access to the **REST API and MongoDB database**.

3. Intrusion Detection and Prevention (IDS/IPS)

- Using **Snort or Suricata**, pfSense continuously monitors network traffic for **suspicious activities**, such as brute-force login attempts or malicious payloads.
- Helps in **detecting and blocking** potential cyber threats before they can compromise the IoT infrastructure.

4. Traffic Filtering & Network Segmentation

- pfSense can **isolate IoT devices** (ESP8266) from the main network using **VLANs**, reducing the risk of **lateral attacks** in case one device gets compromised.
- Can **filter and restrict** certain types of traffic (e.g., only allow **HTTPS and MQTT**, blocking unsafe protocols).

5. Logging & Real-time Monitoring

- pfSense provides **detailed logs** of all network activity, allowing quick identification of **unusual traffic patterns** or unauthorized access attempts.
- Helps in **troubleshooting security issues** and ensuring compliance with best practices for IoT security.

6. Protection for MongoDB Database

- Blocks direct access to MongoDB from the public internet, allowing connections only from the **trusted API server**.
- Prevents **SQL injection and unauthorized database access** by controlling how data is transmitted and received.

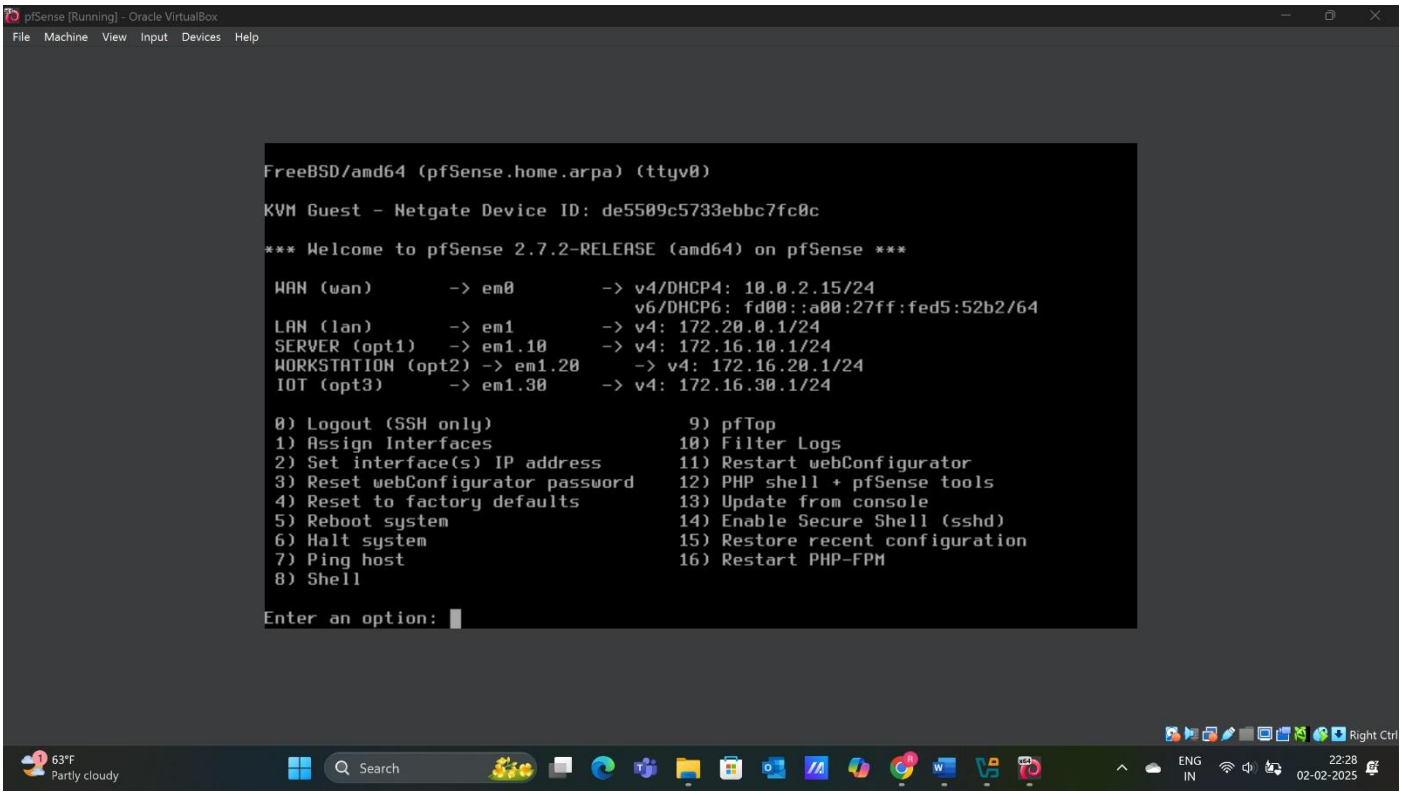

Fig: -pfSense Command Line interface installed in Virtual Box with User specified WAN & LAN address.

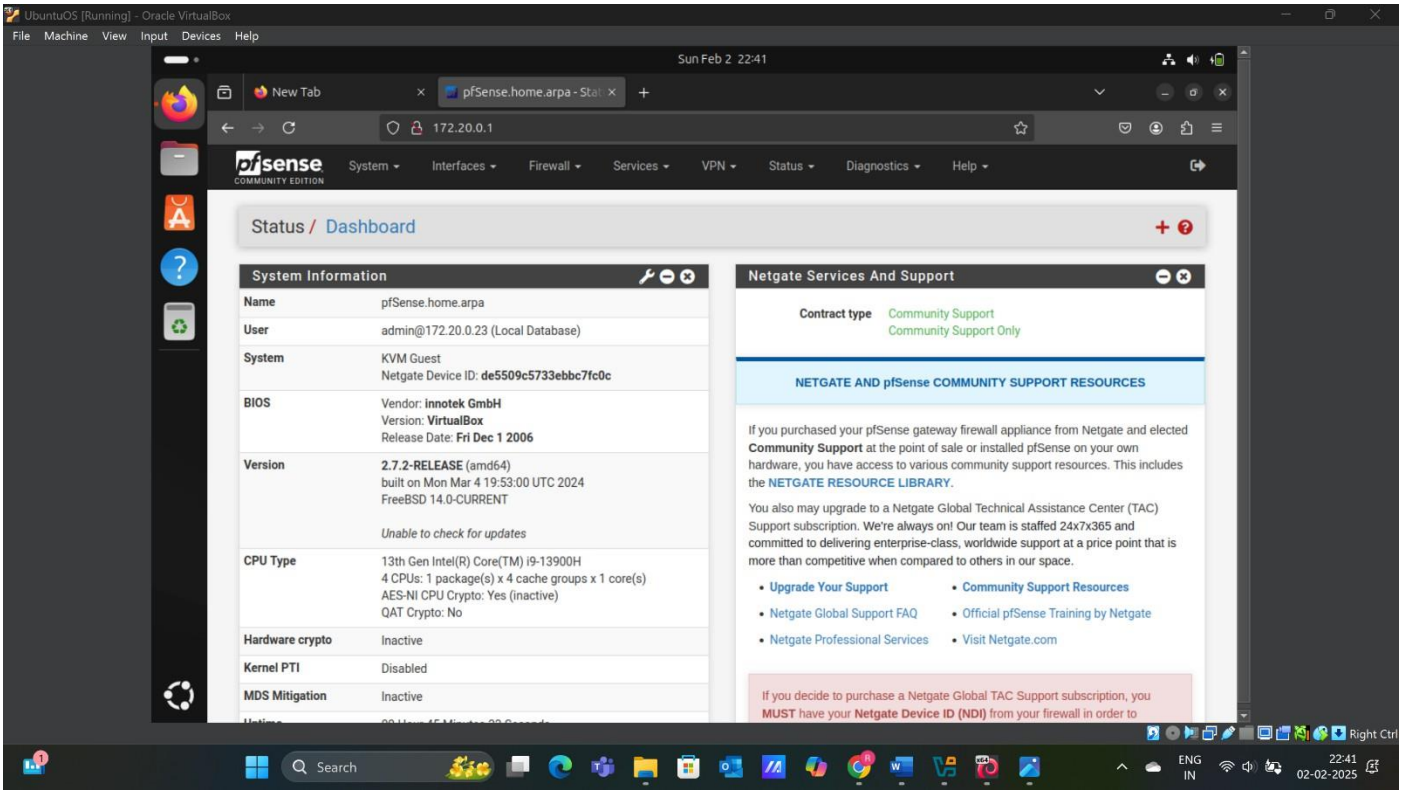

Fig: -pfSense community page where we can configure firewall rules manually

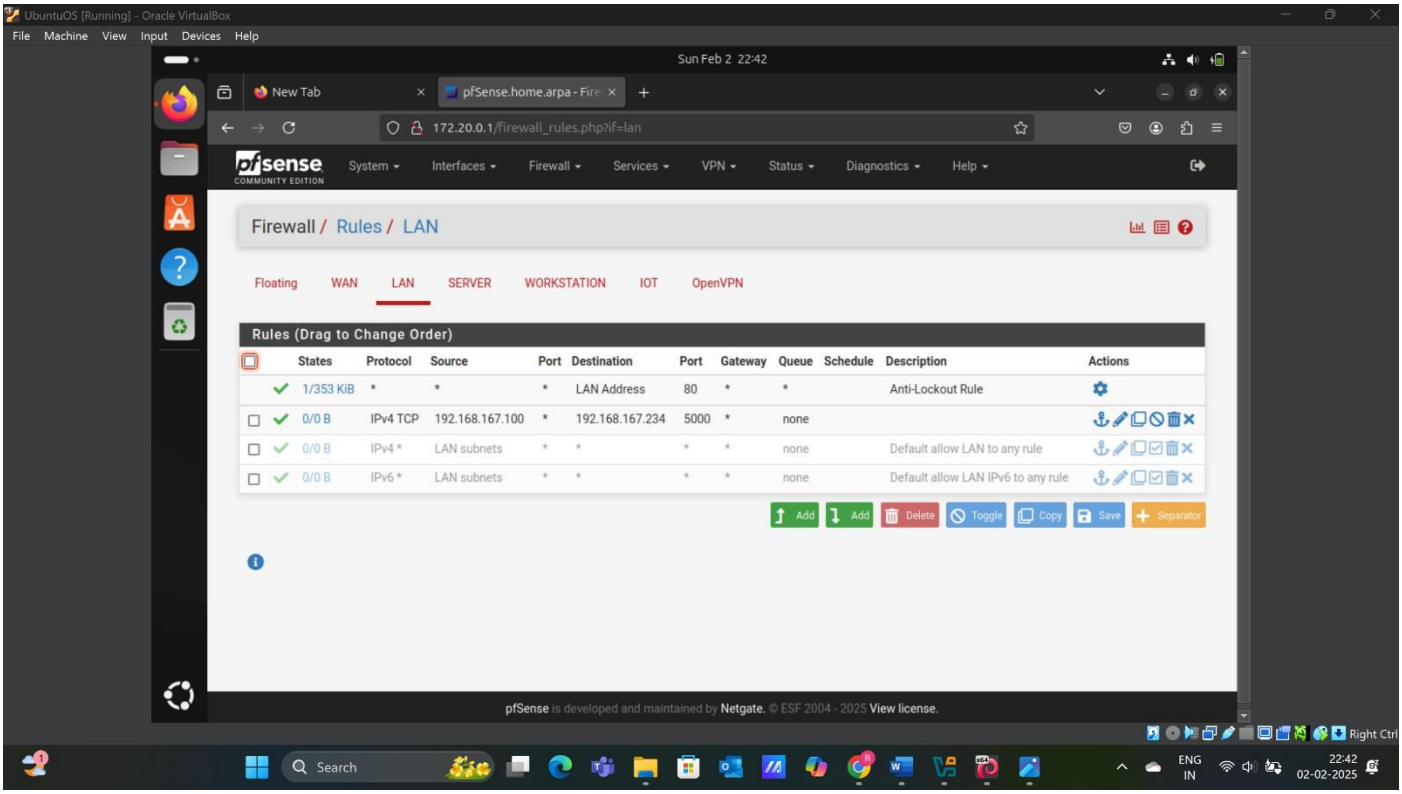

Fig: -Firewall rules set manually for LAN traffic, for the connection between Arduino IDE and Rest API server

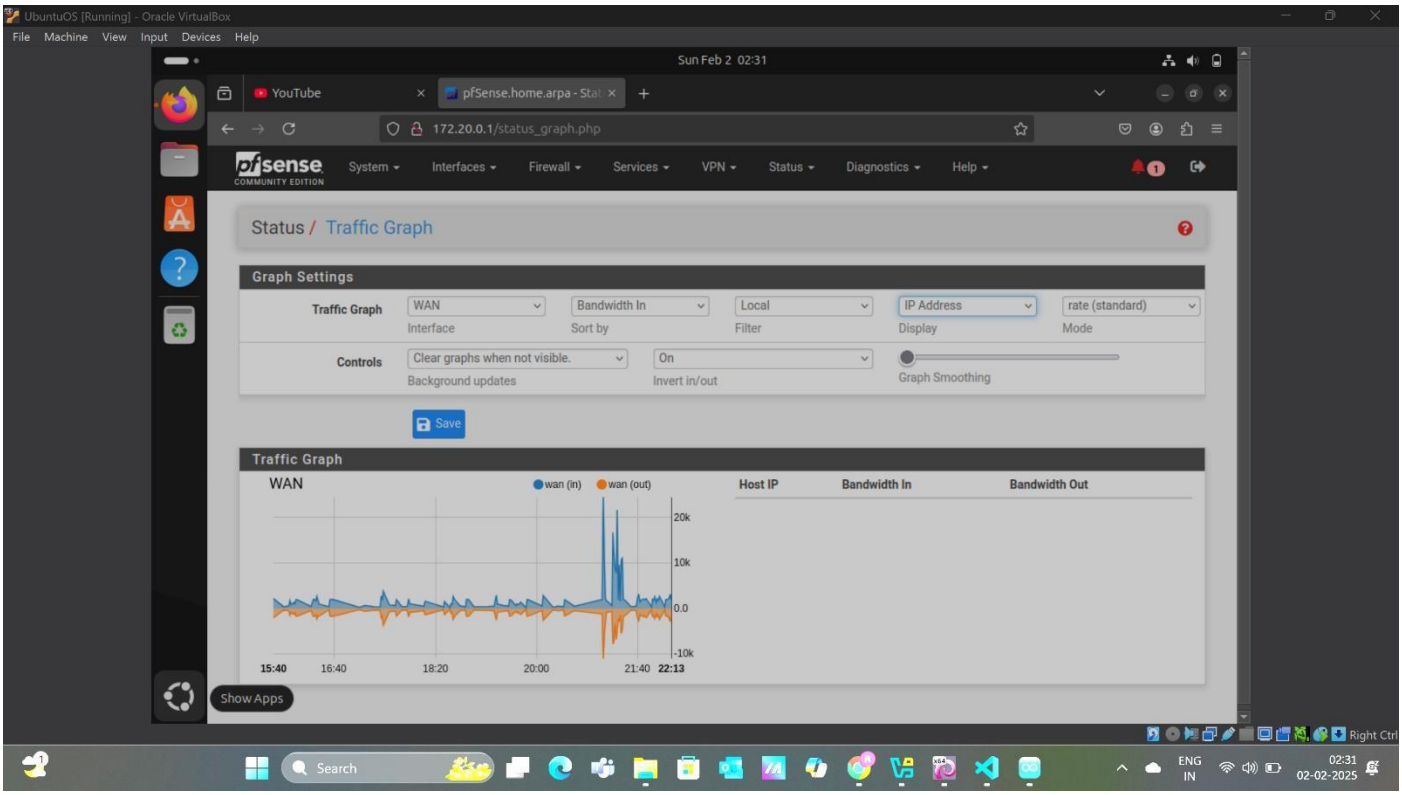

Fig: -Traffic Graph after the connection made

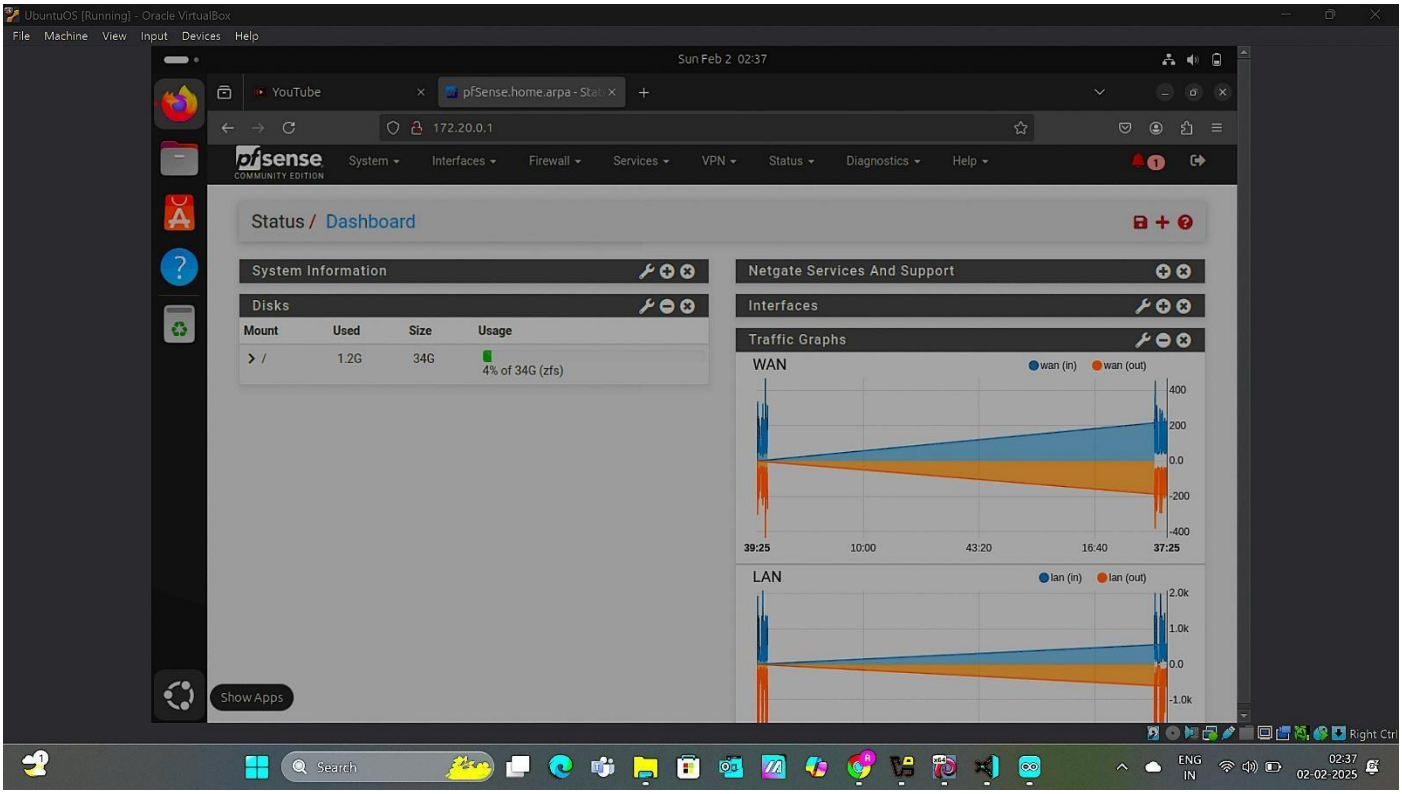

Fig: -Network traffic status for both WAN and LAN interfaces

8 Conclusion

This project successfully integrates IoT devices with RESTful API and MongoDB for real-time sensor data storage and analysis. The ESP8266 microcontroller and DHT11 sensor capture accurate temperature, humidity, and heat index measurements, which are transmitted via Wi-Fi and stored in a scalable MongoDB Atlas database. The modular and cost-effective design ensures high data accuracy and transmission reliability, making the system suitable for smart home monitoring and industrial automation. Extensive testing confirms the system's robustness, and the collected data is visualized using ThingSpeak, providing real-time analytics to enhance usability.

The incorporation of pfSense strengthens the project's security, offering protection against unauthorized access and cyber threats through stateful packet inspection, VPN support, and traffic shaping. This integrated approach addresses current sensor networking challenges while laying a foundation for future enhancements such as real-time analytics and automated alerts. The combination of advanced hardware, secure network infrastructure, and scalable data management establishes a reliable and efficient environmental monitoring solution.