Environmental Monitoring System Based on AWS IoT and Web Dashboard

Author: [Yunfeng Li]

Course: [158335 The Internet of Things and Cloud Computing]

Date: [08/06/2025]

1. Introduction

This project addresses the need for real-time environmental monitoring (temperature, humidity, and pressure) using low-cost devices and cloud integration. It is designed to offer remote visualization, threshold-based alerting, and secure user interaction.

2. Problem Identification

Environmental monitoring is critical for detecting conditions that may impact human health or equipment. Traditional solutions are expensive and non-interactive. This project uses a Raspberry Pi and AWS IoT Core to build a cost-effective system with real-time alerts, Interactive features and web visualization.At the same time, this solution constructs a secure login access to ensure information security.

3. System Architecture

The system includes:

- Sensor Node (Raspberry Pi + Sense HAT Emulator): Simulates data collection.

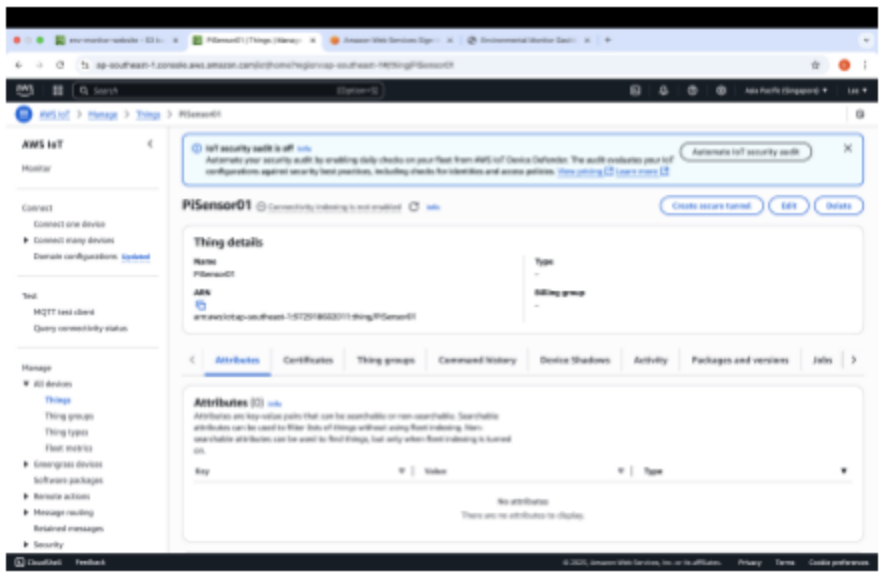
The sensor_writer code is used to log sensor data.

```
import time
import json
from sense_emu import SenseHat

sense = SenseHat()

while True:
    temp = round(sense.get_temperature(), 2)
    hum = round(sense.get_humidity(), 2)
    pres = round(sense.get_pressure(), 2)
    timestamp = int(time.time() * 1000)

    payload = {
        "device_id": "raspi-01",
        "timestamp": timestamp,
        "temperature": temp,
        "humidity": hum,
        "pressure": pres
    }

    with open("/tmp/sense_data.json", "w") as f:
        json.dump(payload, f)

    print("Wrote:", payload)
    time.sleep(5)
```

The aws_uploader code is used to upload the recorded data to AWS IoT.

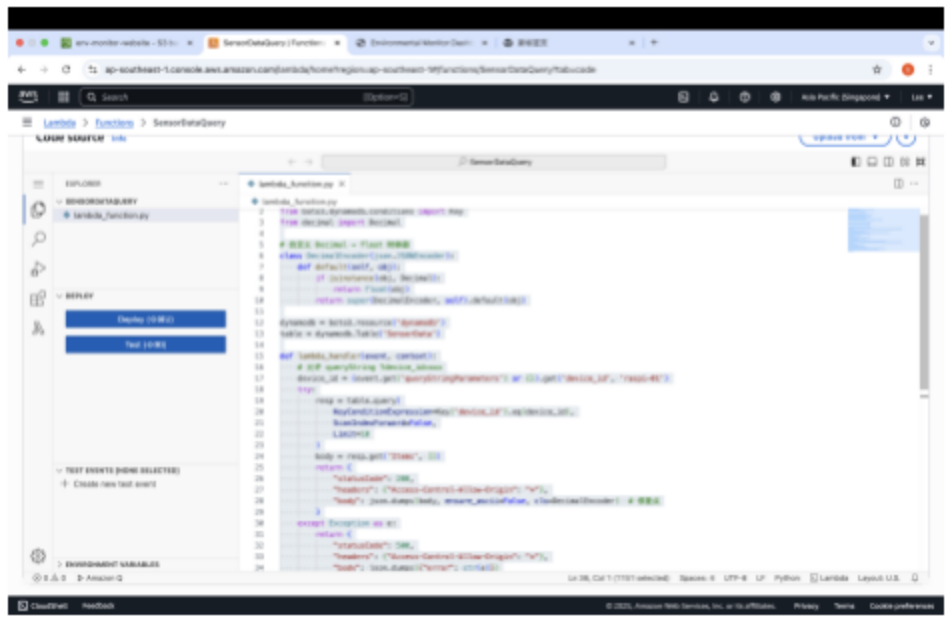
```
import time
import json
import os
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

ENDPOINT = "a9elbwkppvwl9-ats.iot.ap-southeast-1.amazonaws.com"
PORT = 8883
CLIENT_ID = "raspi-client"
TOPIC = "sensors/data"

CA_PATH = "/home/lyfbzcl/iot-home-monitor/certs/AmazonRootCA1.pem"
KEY_PATH = "/home/lyfbzcl/iot-home-monitor/certs/private.pem.key"
CERT_PATH = "/home/lyfbzcl/iot-home-monitor/certs/device.pem.crt"

client = AWSIoTMQTTClient(CLIENT_ID)
client.configureEndpoint(ENDPOINT, PORT)
client.configureCredentials(CA_PATH, KEY_PATH, CERT_PATH)
client.configureConnectDisconnectTimeout(10)
client.configureMQTTOperationTimeout(5)
client.configureAutoReconnectBackoffTime(1, 32, 20)
client.connect()
print("Connected to AWS IoT Core")

while True:
    if os.path.exists("/tmp/sense_data.json"):
        with open("/tmp/sense_data.json", "r") as f:
            payload = json.load(f)
        client.publish(TOPIC, json.dumps(payload), 1)
        print("Published:", payload)
    time.sleep(5)
```

– AWS IoT Core: Receives MQTT data.

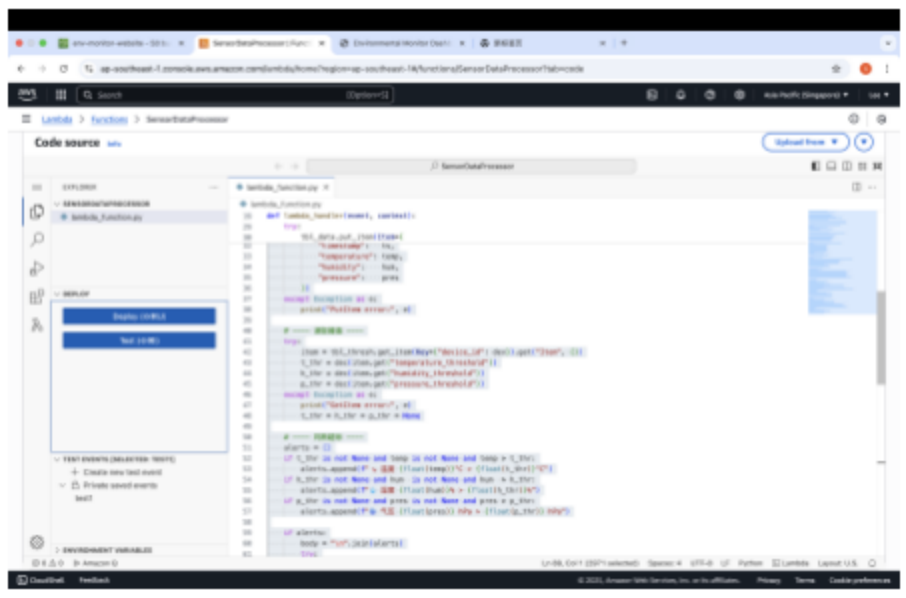


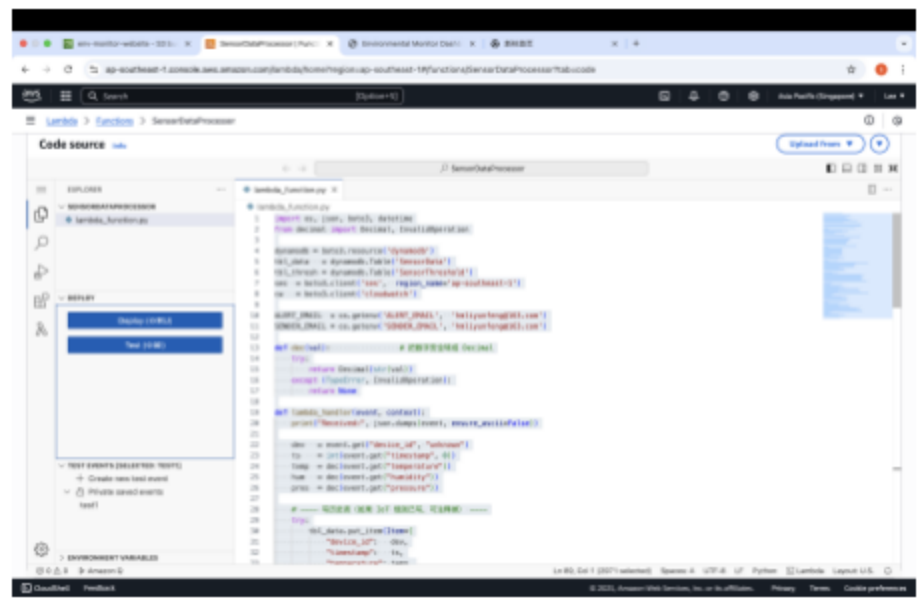
– AWS Lambda: Processes data (e.g., threshold alerting via email).

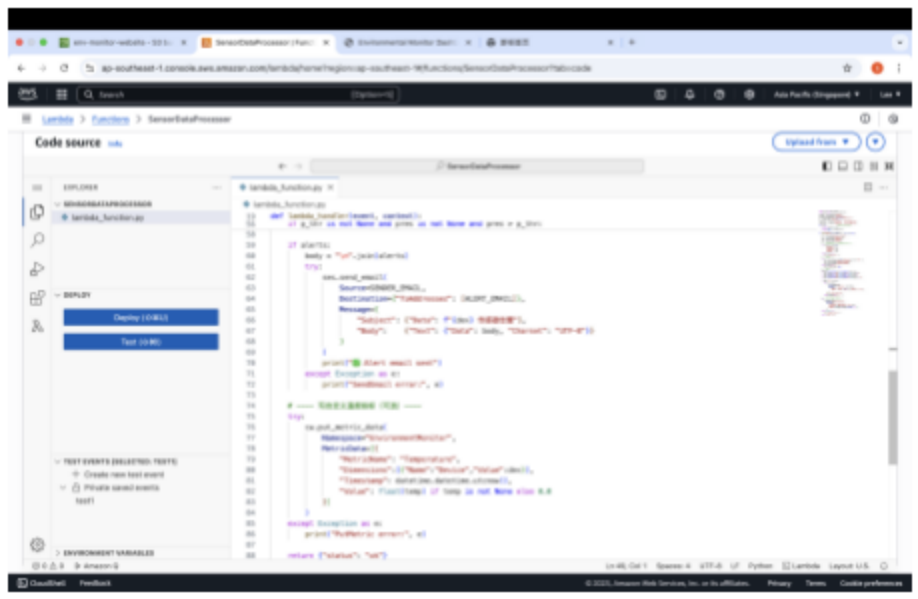
SensorDataQuery is used to read the latest sensor data from the DynamoDB table SensorData.



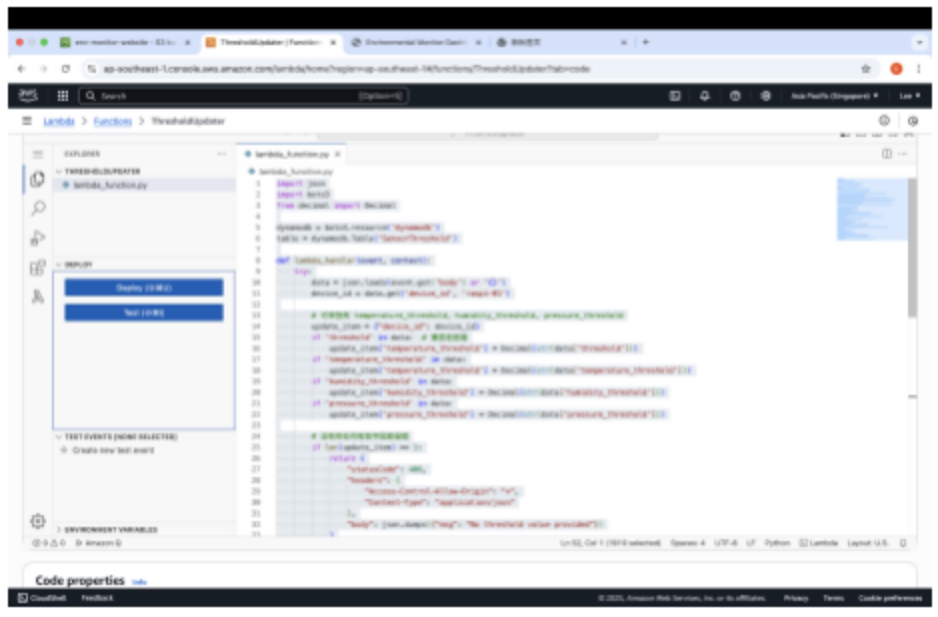
SensorDataProcessor is triggered when the device uploads data to AWS IoT Core, reads the corresponding threshold setting (from the SensorThreshold table), compares the current data to the threshold, and if it does, sends an alert email (via Amazon SES).

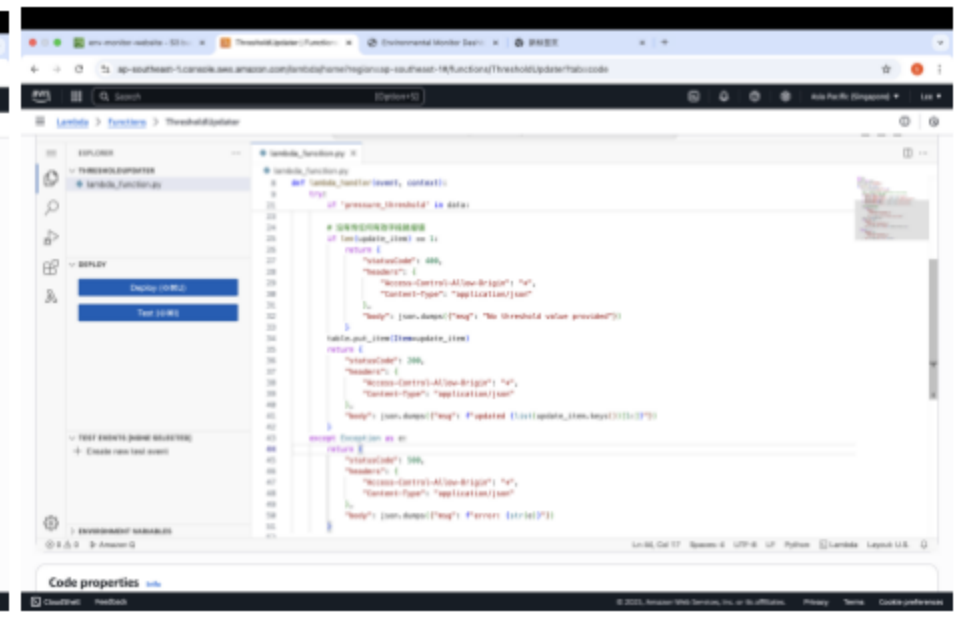




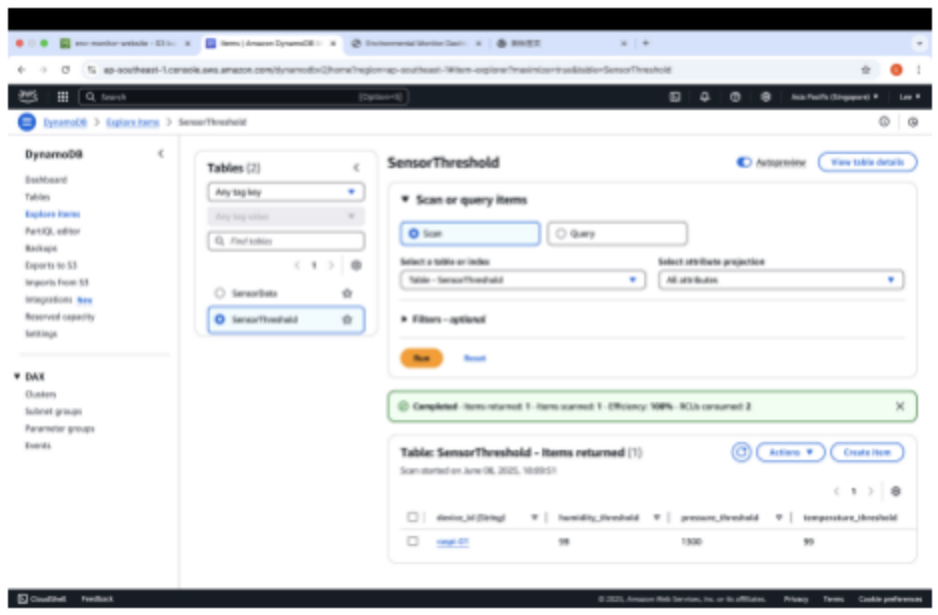


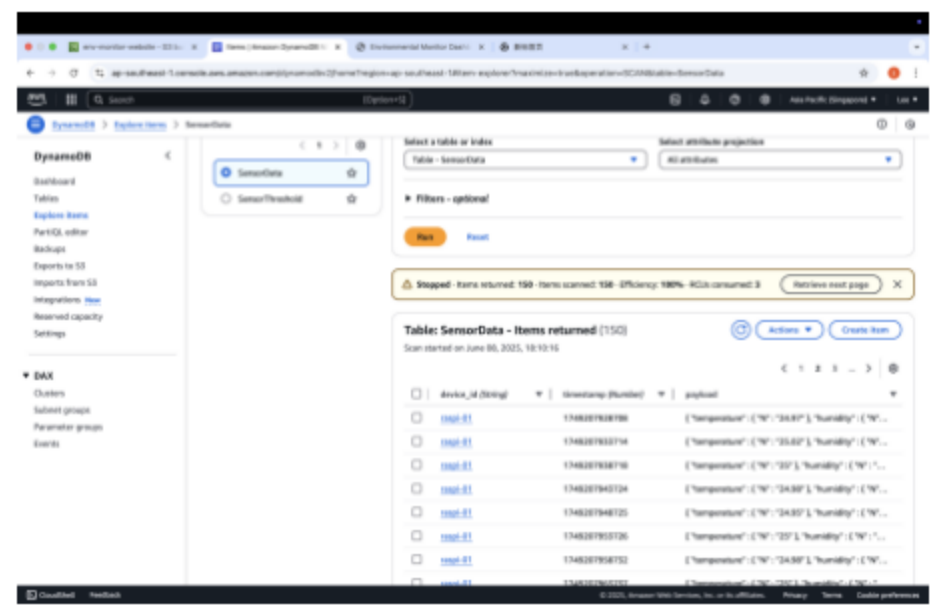
ThresholdUpdater accepts threshold Settings (e.g., temperature, humidity, air pressure) submitted by the front end and writes or updates them to the SensorThreshold table.





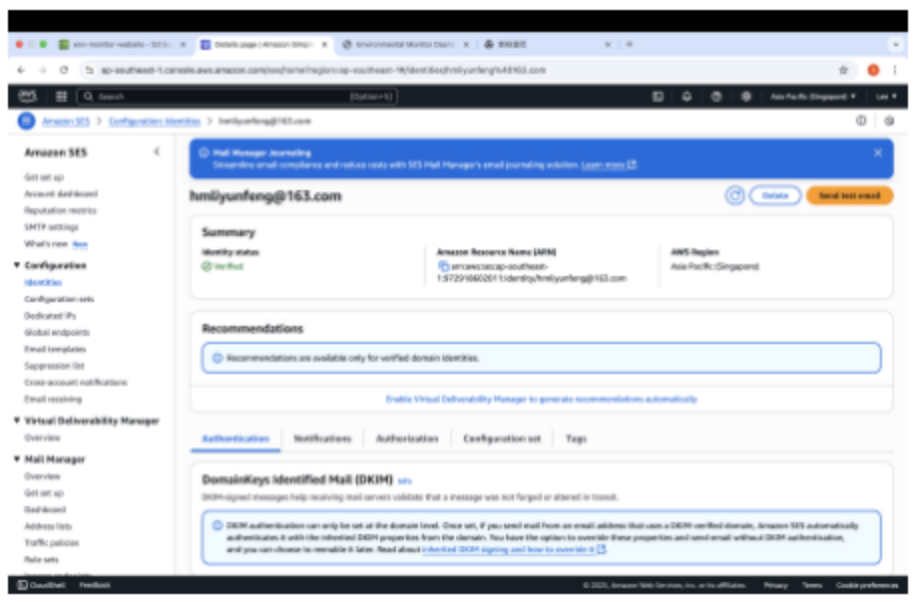
– DynamoDB: Stores environmental readings and thresholds.



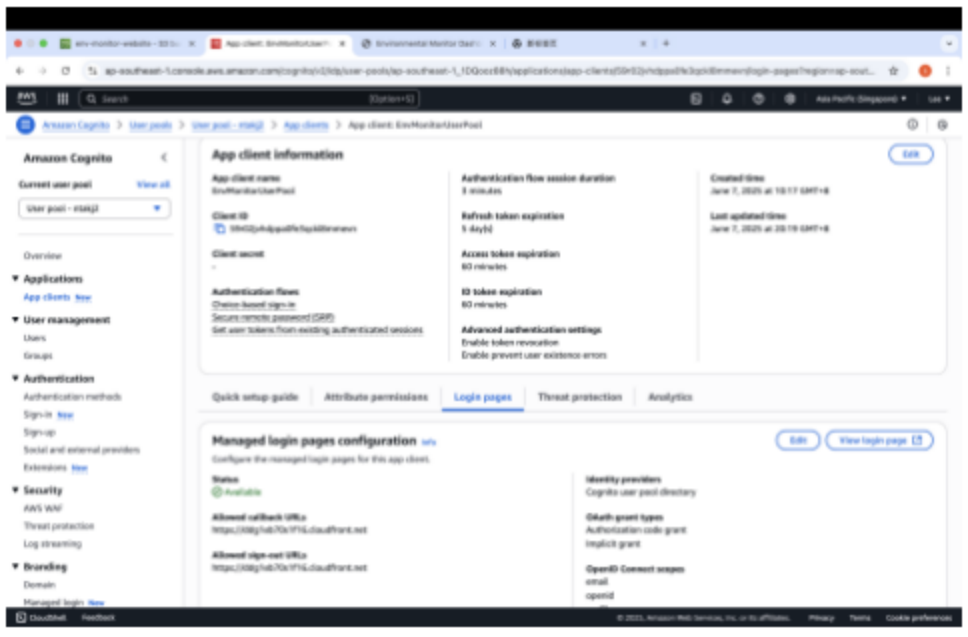


– Amazon SES: Sends email alerts.

Only send alert emails to verified email addresses



– Amazon Cognito: Handles secure user login.



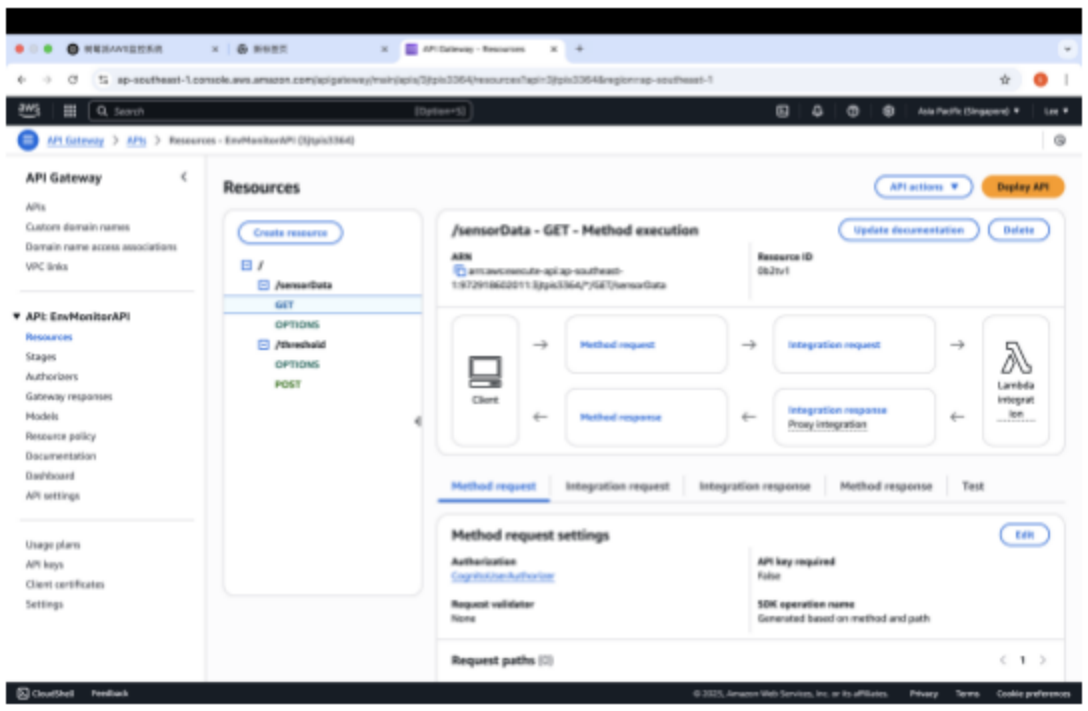
– CloudFront + S3: Hosts frontend interface.

The code for the web page is in the attachment index.txt

– JavaScript Web Dashboard: Visualizes data and supports threshold configuration.

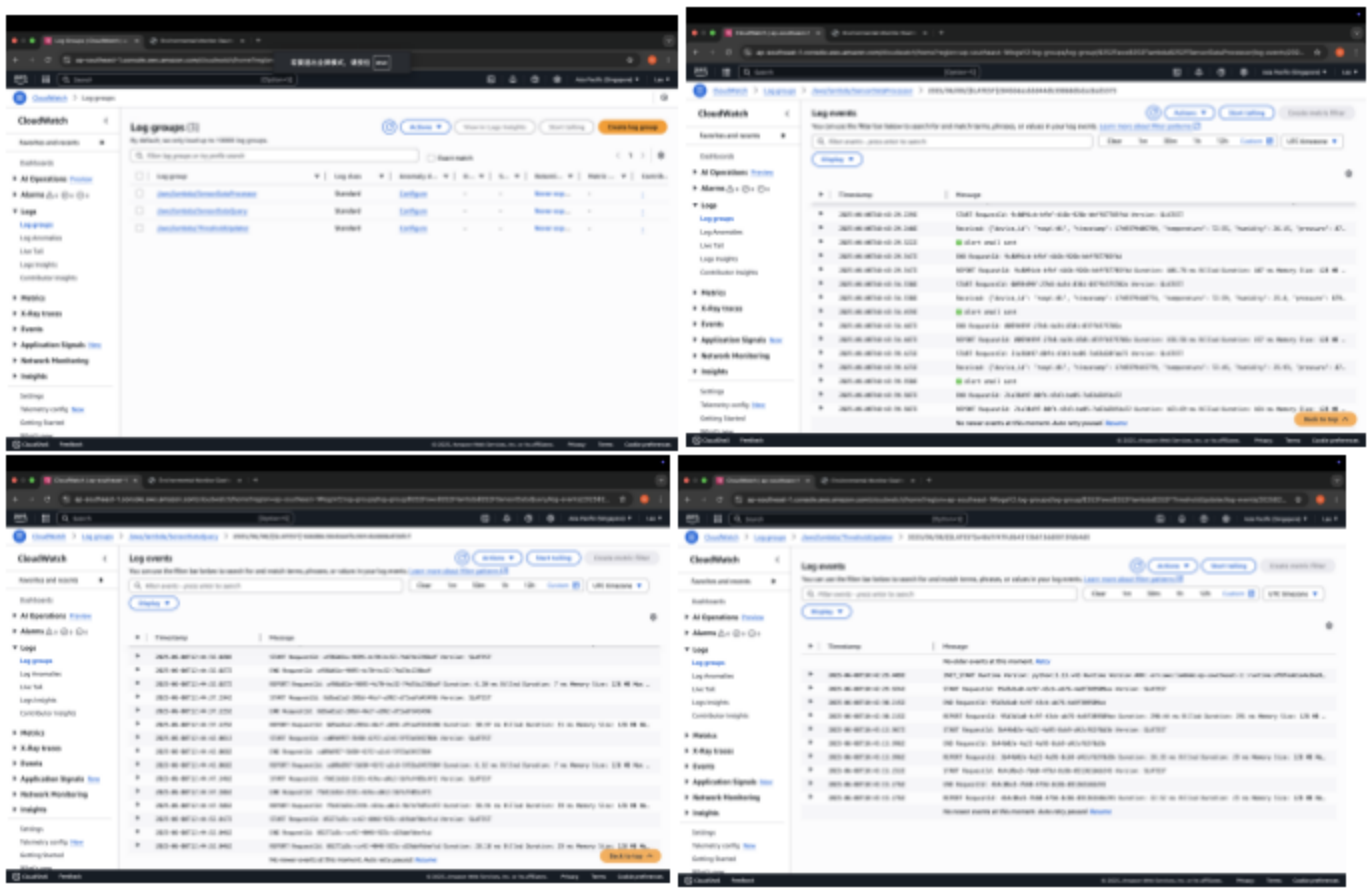
The code for the web page is in the attachment index.txt

– API Gateway



–Cloud Watch

see all lambda data here

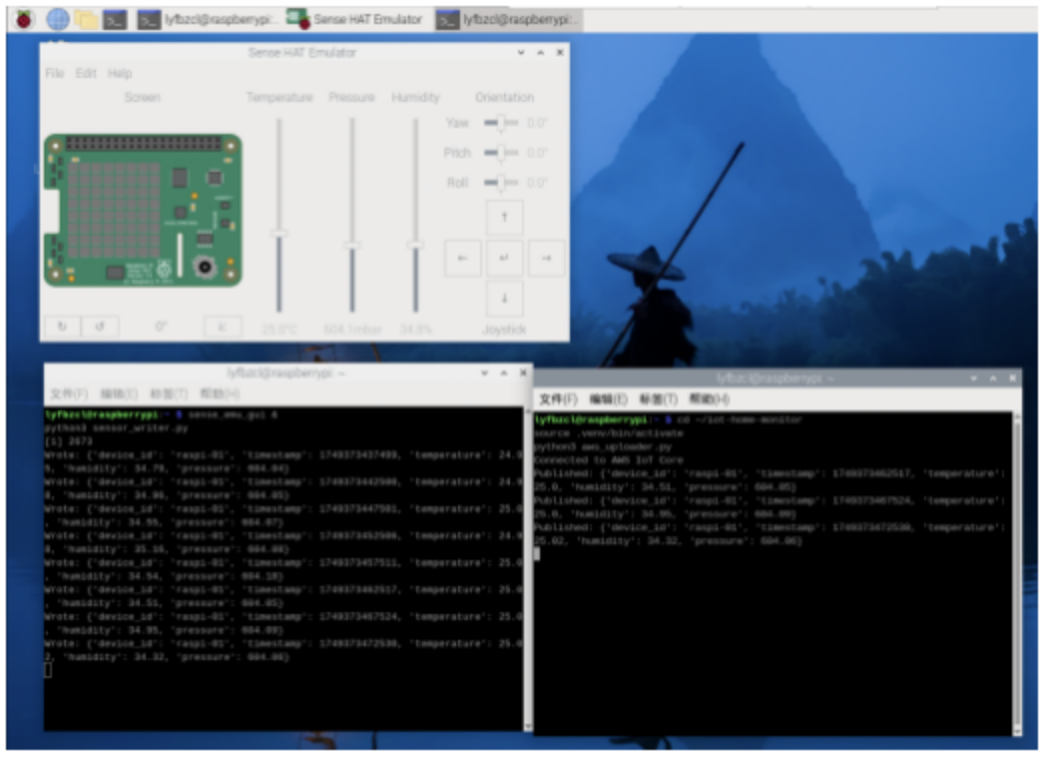


–IAM

4. Prototype Implementation

4.1 Device Scripts

– sensor_writer.py and aws_uploader.py simulate Sense HAT readings and publish via MQTT to AWS IoT.

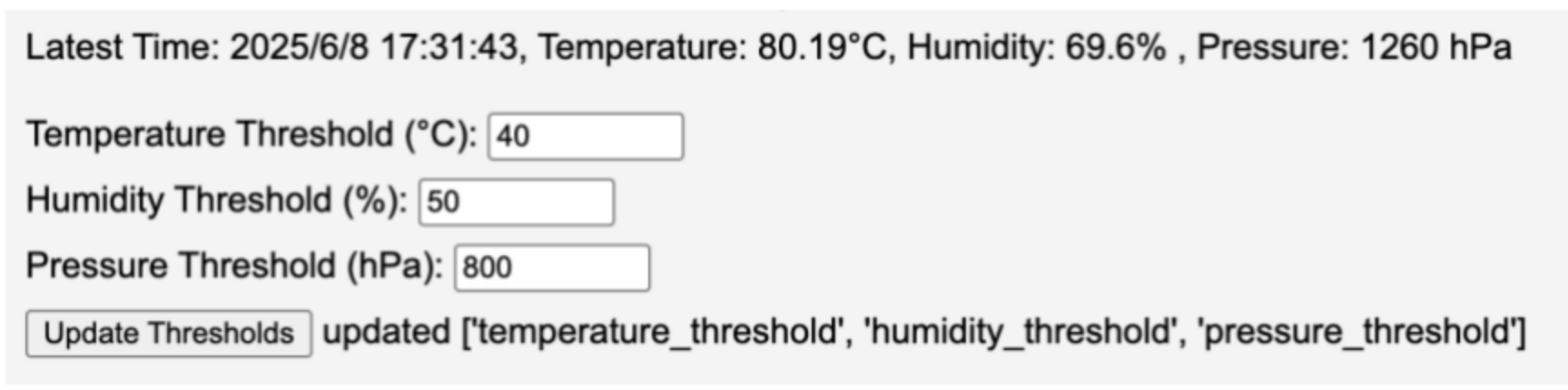


– Device ID used: raspi-01.

4.2 Lambda Functions

– SensorDataProcessor: Compares readings to thresholds in SensorThreshold, sends alerts via SES.

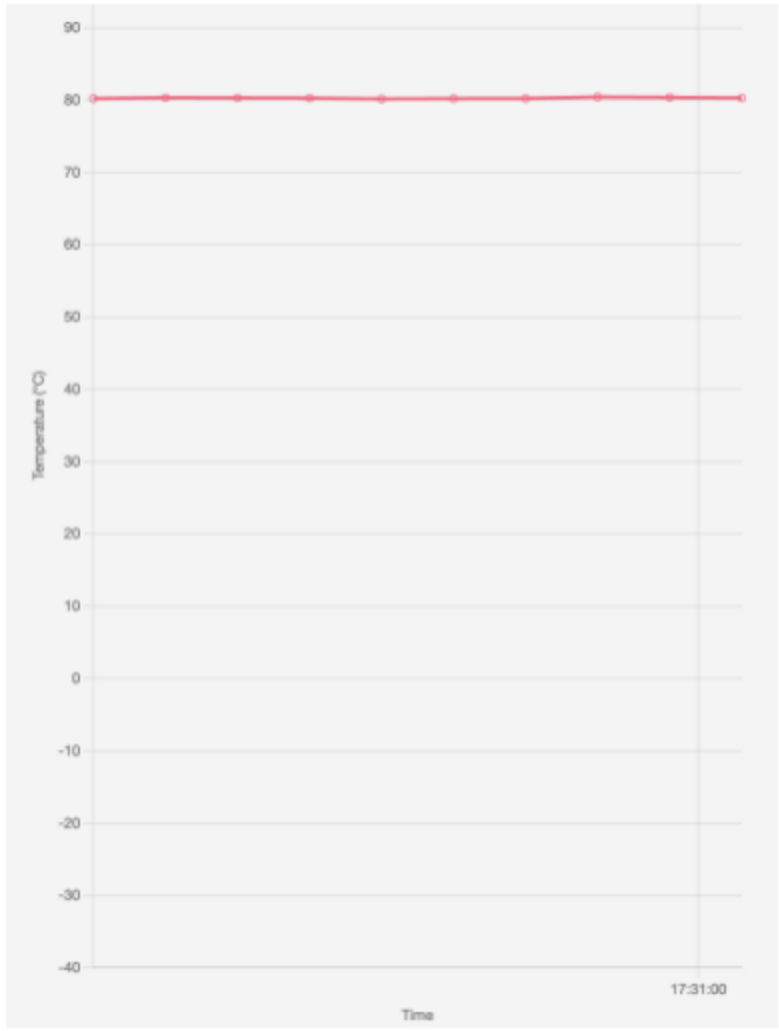
– ThresholdUpdater: Saves updated threshold settings from frontend.

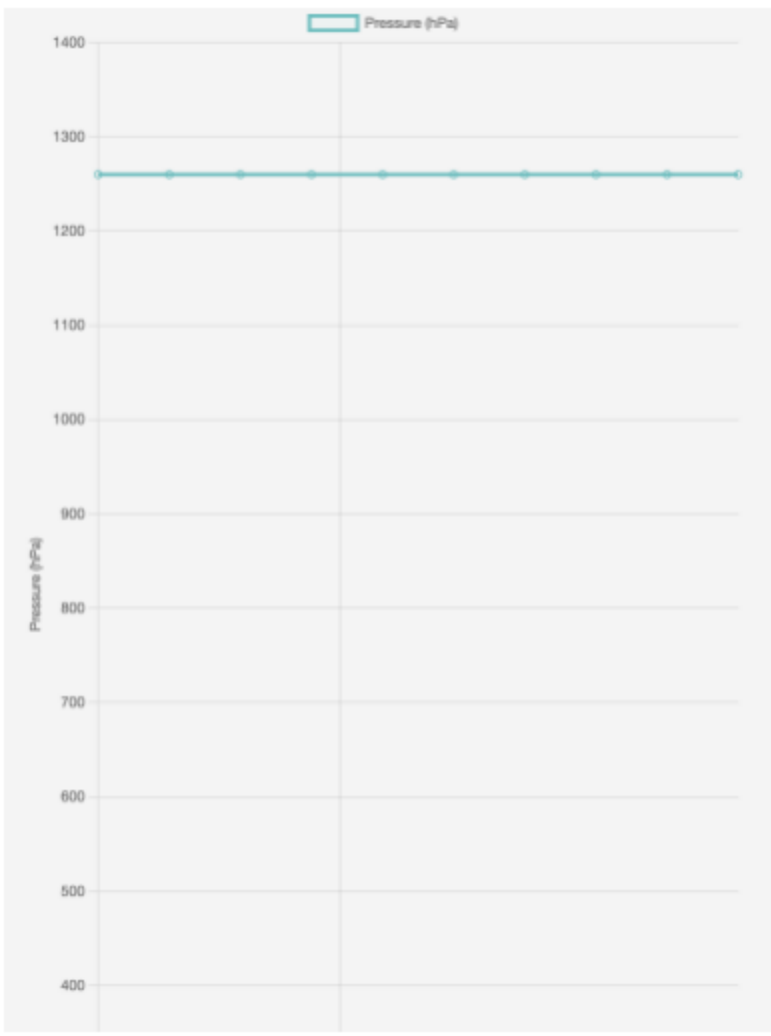


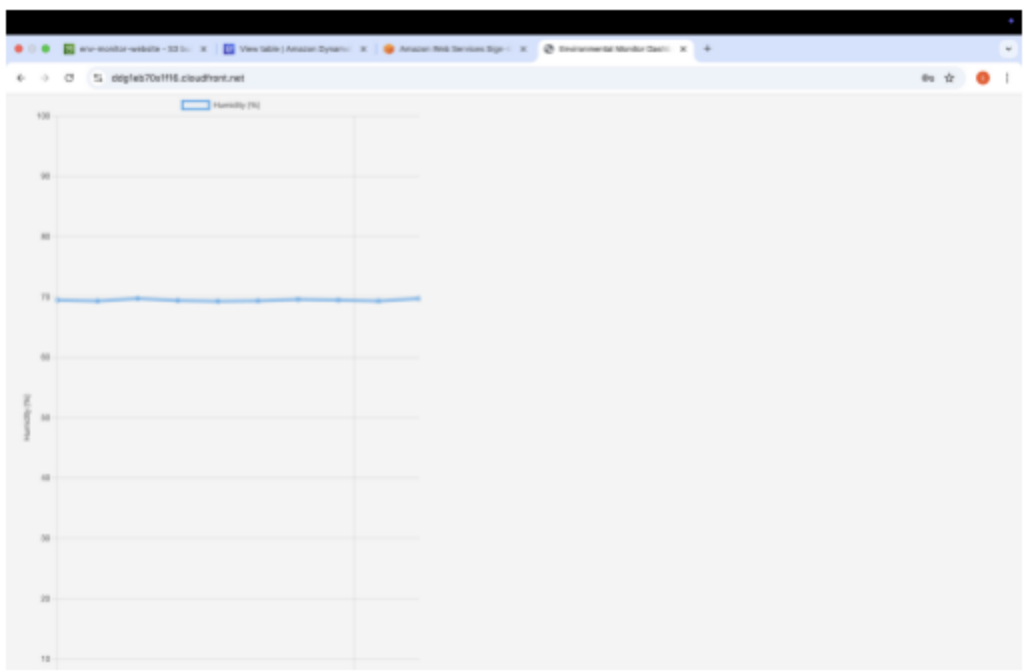
– SensorDataQuery: Returns recent data for plotting.

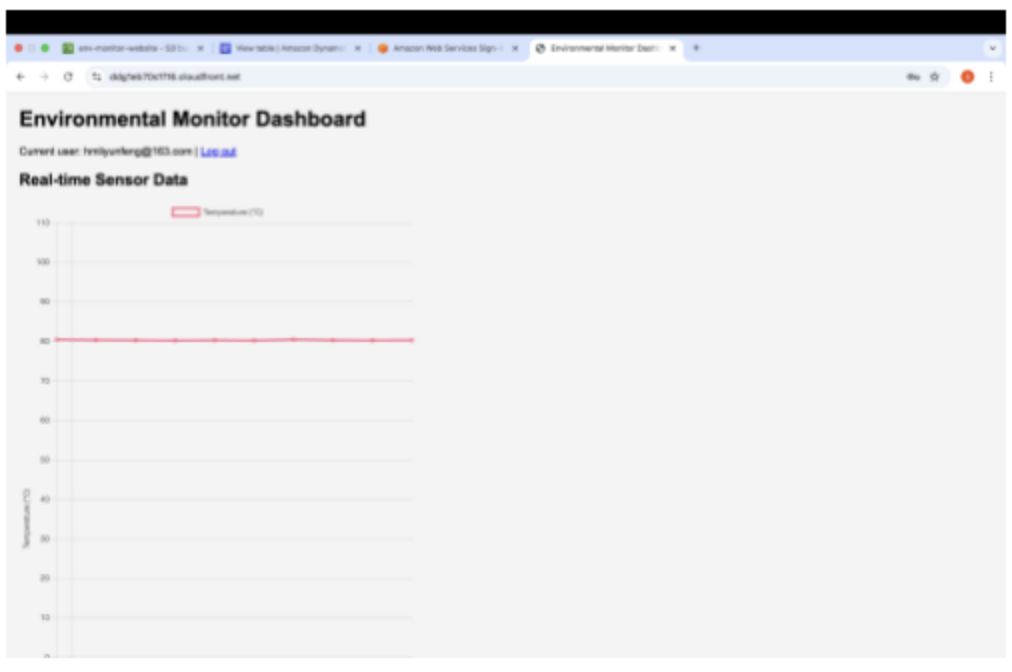
4.3 Frontend Dashboard (index.html)

– Built using HTML, CSS, and JavaScript with Chart.js.



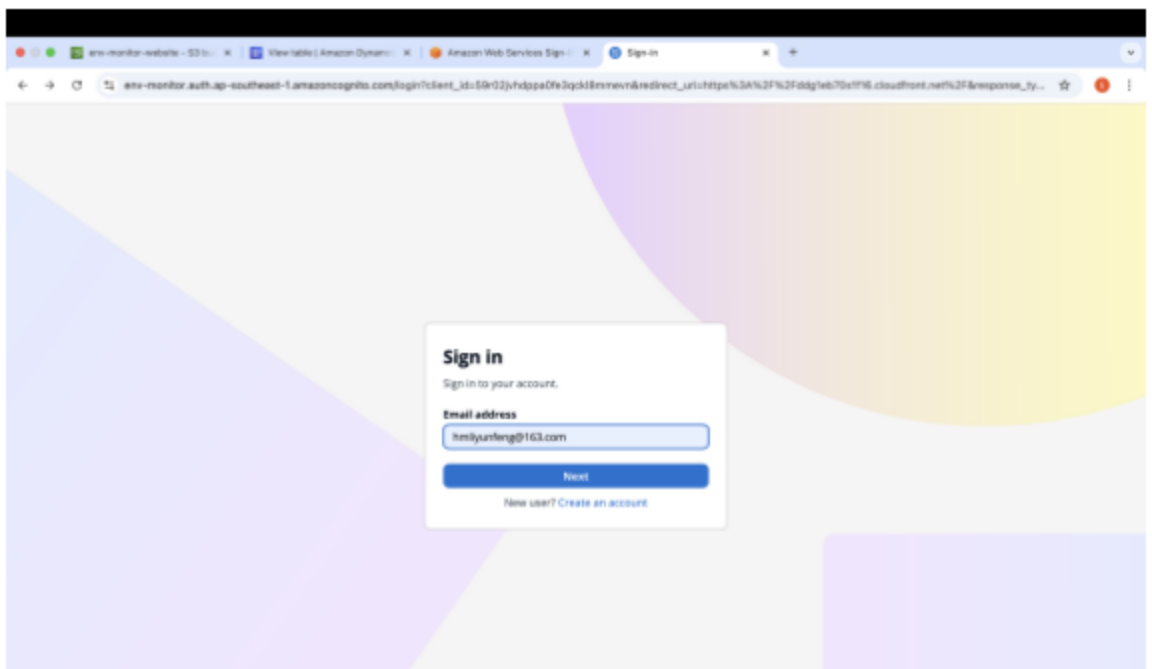


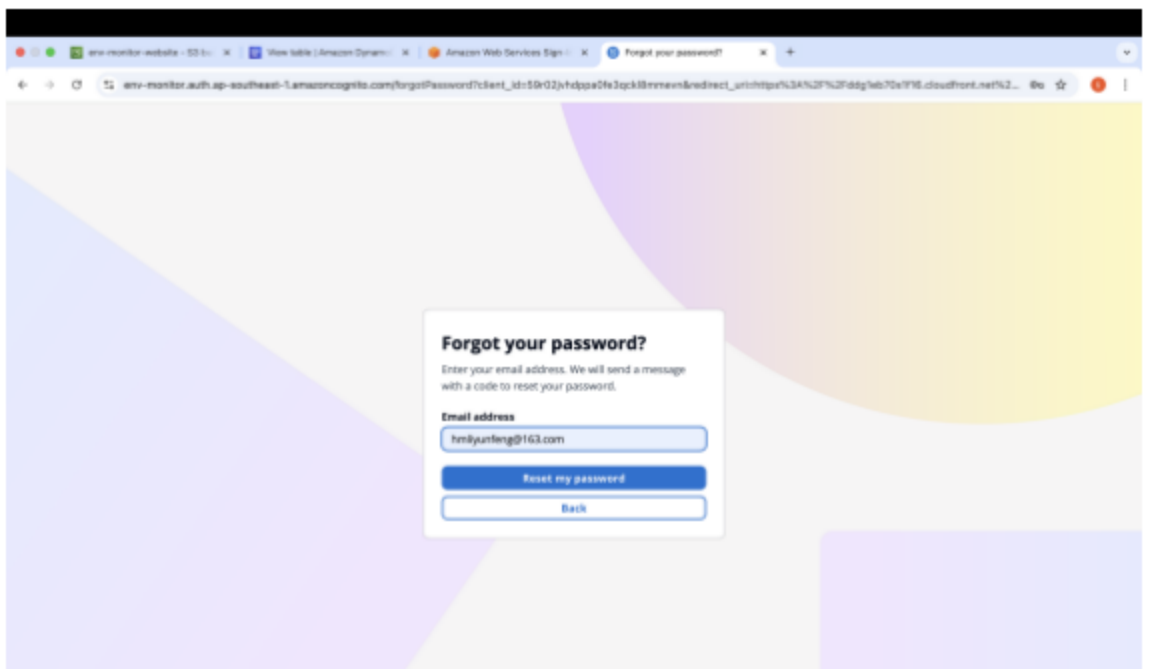


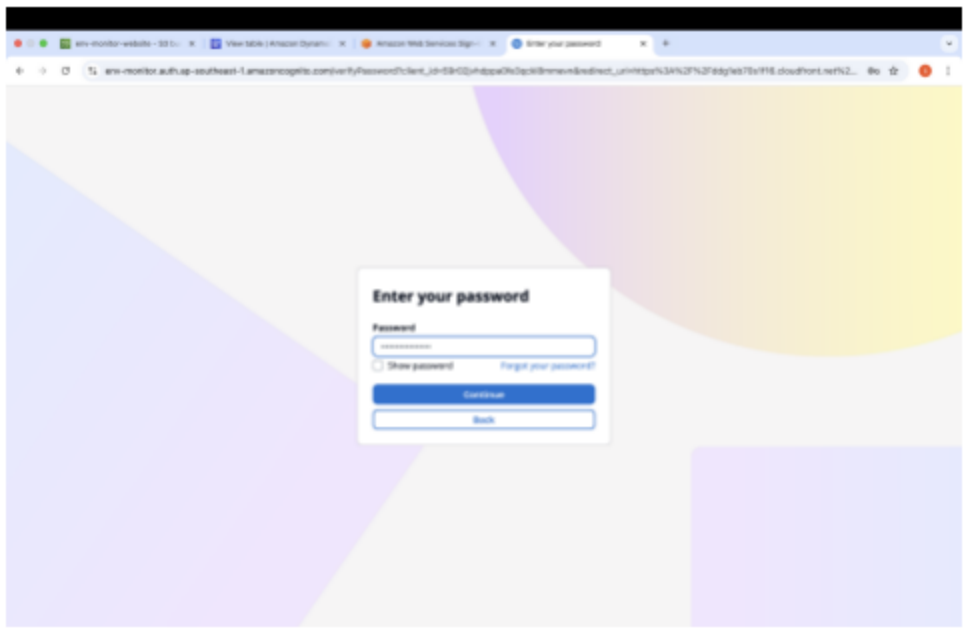


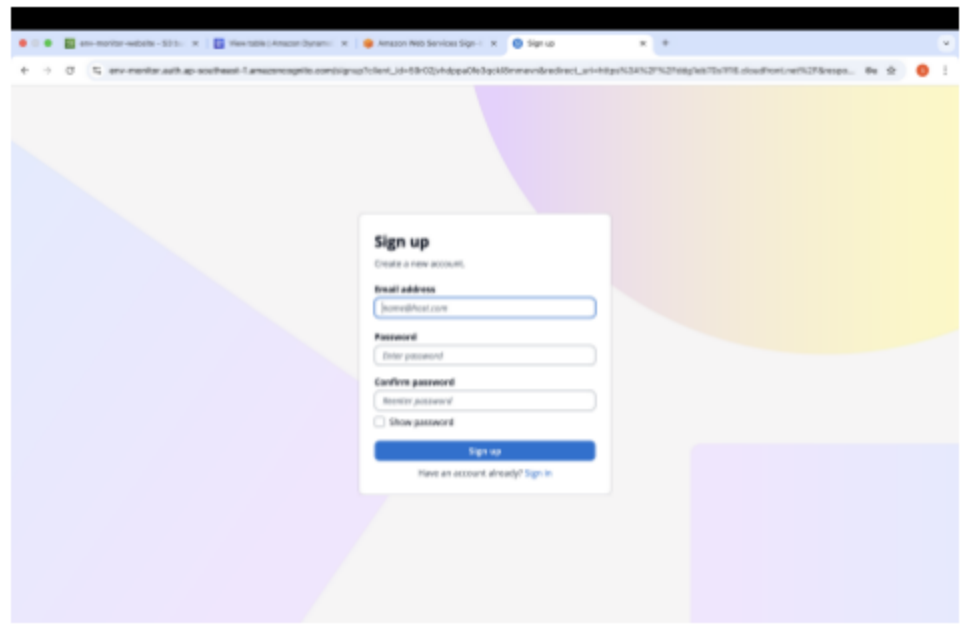
– Features:

- Real–time data plotting (temperature, humidity, pressure).
- Secure login/logout using Cognito.
- Threshold input form auto–filled from backend.
- Threshold update with immediate feedback.









5. Key Features and Justifications

Feature	Justification
Web-based dashboard	Platform-independent, easy to deploy.
Threshold-based alerting	Allows proactive intervention.
Serverless AWS architecture	Scalable and cost-efficient.
Secure login with Cognito	Ensures user data protection.

Deployment methods

Deployment: Run on Raspberry PI terminal 1

sense_emu_gui &

python3 sensor_writer.py

Run on Raspberry PI Terminal 2

```
cd ~/iot-home-monitor

source .venv/bin/activate

python3 aws_uploader.py
```

In the web page to https://ddg1eb70s1f16.cloudfront.net/ to log in operation and subsequent pages threshold Settings

Bonus – Data Analytics (Optional +5 Marks)

The system incorporates fundamental data analytics functionality to enhance user experience and support environmental monitoring decisions. While this was an optional task, several components of the system demonstrate analytics capabilities:

1. Real-Time Time-Series Visualization

All sensor readings (temperature, humidity, pressure) are stored with timestamps in a structured format within the DynamoDB `SensorData` table. These values are fetched and visualized on the dashboard using JavaScript and Chart.js, producing dynamic line charts that display trends and fluctuations over time. This enables users to identify anomalies, patterns, and environmental changes at a glance.

2. Threshold-Based Event Detection

The `SensorDataProcessor` Lambda function performs conditional analytics: it compares incoming sensor data against threshold values stored in the `SensorThreshold` table. If a value exceeds the user-defined threshold, it triggers an alert via Amazon SES. This rule-based analysis layer enables automatic detection of critical conditions.

3. User-Controlled Data Parameters

Users can define and update custom thresholds for temperature, humidity, and pressure via the dashboard. These thresholds are stored in the cloud and directly affect the logic of real-time data evaluation. This interactivity makes the analysis process dynamic and user-driven.

4. Decision Support via Visual Feedback

The visual trends and alerting mechanism provide users with actionable insights. By reviewing charts and receiving real-time alerts, users can make timely decisions—such as adjusting indoor conditions or investigating sensor anomalies.

> These combined features demonstrate a complete pipeline from data collection to cloud storage, visualization, threshold-based analysis, and feedback delivery—effectively satisfying the bonus criteria for data analytics.

6. Challenges & Solutions

– Issue: No physical sensor on Pi.

Solution: Used sense_emu to simulate.

– Issue: Thresholds not auto-filled initially.

Solution: Added fetchAndFillThresholds() in frontend.

– Issue: Lambda not triggering alert.

Solution: Corrected field names and ensured SES configuration.

7. Future Work

– Add charts for historical trends.

– Use real sensor hardware for deployment.

– Extend to more IoT devices (multi-device support).

– Include SMS alerts via SNS.

8. Conclusion

This project demonstrates a fully functional IoT-cloud system with secure interaction, real-time visualization, and configurable alerts. It meets all functional goals and exemplifies good system design using AWS.