Smart Green House with AWS Cloud Services and IoT – Mohammed Ali 249355 – Mohammed Alramadan 242344

Introduction:

In today's technological landscape, IoT and Cloud/Edge computing are revolutionizing environmental monitoring. Our project focuses on measuring temperature and humidity using the DHT11 sensor and ESP32 microcontroller, transmitting data via MQTT to an Amazon EC2 server through VPC.

This server, running on Ubuntu, utilizes the Mosquitto MQTT broker and Node-RED for data management. The motivation behind this project is the need for real-time environmental monitoring to enhance agricultural productivity, and smart greenhouse monitoring and control.

Accurate temperature and humidity data improve decision-making in these areas. AWS services play a crucial role in our project. Data is processed and stored using AWS DynamoDB, while AWS Amplify and CodeCommit manage the web service, with code hosted on S3. Continuous deployment ensures seamless updates. AWS Cognito with Lambda handles authentication, ensuring only University of Calabria students can access the system.

Our web server displays the maximum and minimum temperatures and humidity levels and the current values besides the threshold in which a notification will happen, and Node-RED sends email notifications when thresholds are exceeded. An additional ESP32 provides auditory alerts via a voice alarm. This project demonstrates the effective use of IoT and Cloud/Edge computing to create a robust, scalable environmental monitoring system, offering real-time data, efficient solutions for various applications.

Analysis

Constraints and Characteristics

Scenario Constraints:

Environment: Indoor monitoring of temperature and humidity in a controlled environment. **Coverage Area:** Limited to the immediate vicinity of the sensors, typically within a single room (greenhouse). **Connectivity:** Reliable Wi-Fi connectivity is available. **Hardware:** Utilizes DHT11 sensors and ESP32 microcontrollers for data acquisition. **Cloud Infrastructure:** Hosted on Amazon EC2, ensuring scalable and flexible cloud resources. **Security:** Data access is restricted to authenticated users from the University of Calabria using AWS Cognito and Lambda.

Functional Requirements:

Real-time Data Monitoring: Continuous measurement of temperature and humidity. **Data Transmission:** Transmit sensor data via MQTT protocol to the cloud server. **Data Storage:** Use AWS DynamoDB functions to store data in a structured format for future retrieval and analysis. **Threshold Alerts:** Set thresholds for temperature and humidity levels and send notifications when these are exceeded. **User Authentication:** Ensure only authorized users can access the data and modify system settings. **Web Dashboard:** Provide a user-friendly web interface for displaying current and historical data, managed by AWS Amplify and CodeCommit. **Email Notifications:** Send email alerts when threshold conditions are met using Node-RED. **Voice Alerts:** Activate a voice alarm on a secondary ESP32 device when thresholds are exceeded.

Non-functional Requirements:

Data Transmission Speed: Ensure data is sent to the server within 5 seconds of measurement. **Notification Delay:** Send email and voice notifications within 5 seconds of detecting threshold breaches. **Security:** Only University of Calabria's email allowed to update thresholds. **Reliability:** Eliminate sensor errors that could happen. **Usability:** The web interface should be intuitive and accessible to users with minimal technical expertise.

High-Level Modelling

SO High-Level Model:

Sensors: DHT11 sensors collect temperature and humidity data. **IoT Devices:** ESP32 microcontrollers process sensor data and transmit it via MQTT. **Gateway:** MQTT broker on Amazon EC2 receives data from ESP32 devices. **Processing Unit:** Node-Red functions process data. **Storage:** Data is stored in a database managed within the AWS.

Notification System: Node-RED handles threshold alerts and email notifications. **User Interface:** AWS Amplify and CodeCommit manage the web dashboard, and AWS Cognito handles user authentication.

Domain Model:

Entities: Sensor Data: Represents temperature and humidity readings. **User:** Authenticated individuals accessing the system. **Threshold:** Preset limits for temperature and humidity triggering alerts. **Notification:** Alerts sent to users when thresholds are breached.

Relationships: Sensor Data is Collected by Sensors, **Sensor Data is Sent by ESP32 to MQTT Broker Sensor, Data is Processed by node-red, Notifications are Triggered by Threshold Breaches**, **Users Receive Notifications**, **Users Access Sensor Data through Web Dashboard or node-red dashboard.**

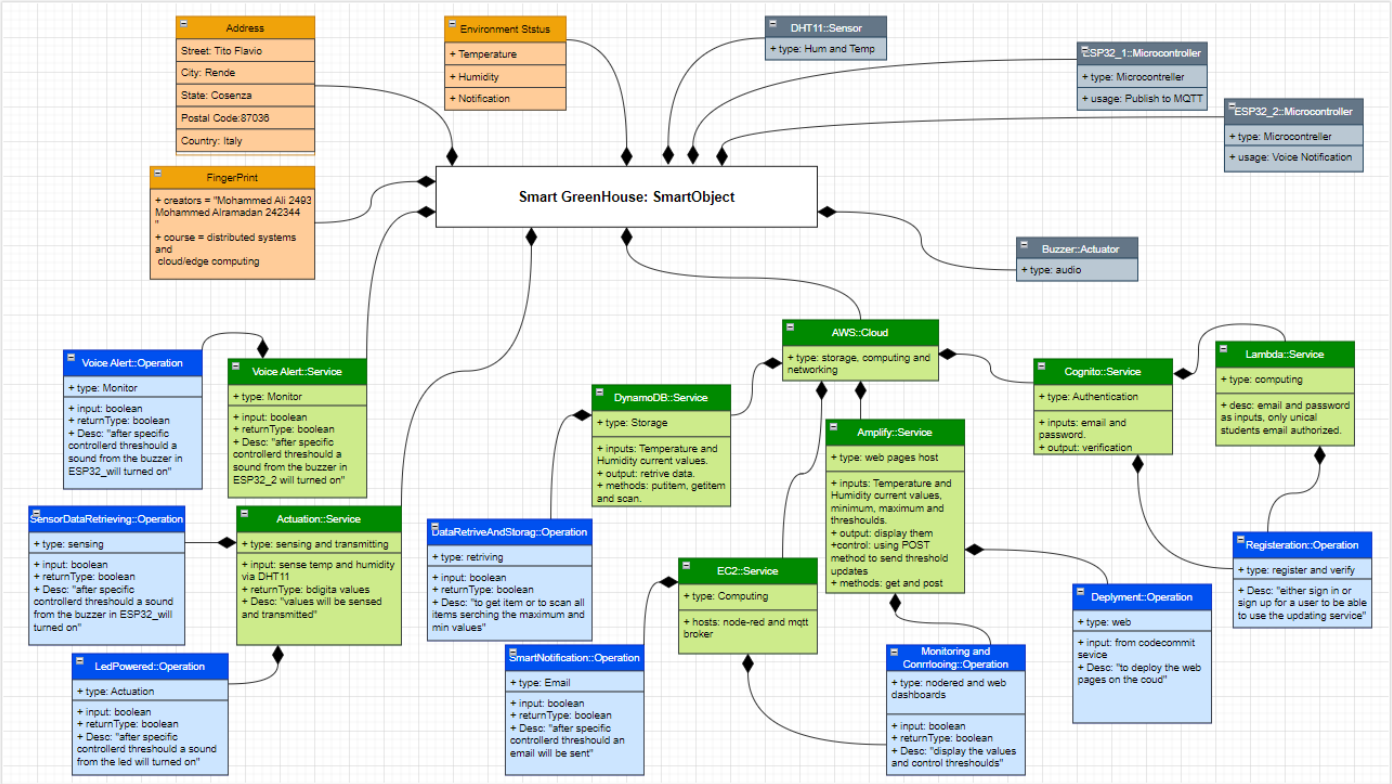

Figure 1: UML State machine

Design

Based on the functional and non-functional requirements outlined in the previous section, we have made several design choices to ensure our system meets the necessary criteria. These design choices focus on the paradigms and general approaches that best align with the requirements of our temperature and humidity monitoring project.

Design Choices

1. Real-time Data Monitoring

- **Design Solution:** Use of MQTT Protocol
- **Justification:** MQTT is a lightweight, publish-subscribe network protocol that is ideal for connections with remote locations where a small code footprint is required or network bandwidth is limited. It ensures efficient real-time data transfer from ESP32 devices to the cloud server.

2. Data Storage

- **Design Solution:** NoSQL Database
- **Justification:** NoSQL databases, such as Amazon DynamoDB, are well-suited for storing time-series data due to their scalability and flexibility. They can handle large volumes of data with ease and provide quick read/write operations, which is essential for real-time applications.

3. Threshold Alerts

- **Design Solution:** Event-driven Architecture
- **Justification:** An event-driven architecture, particularly using Node-Red, allows the system to respond to specific events (e.g., exceeding temperature/humidity thresholds) promptly. This ensures that alerts are generated and processed in real time without unnecessary delays.

4. User Authentication

- **Design Solution:** AWS Cognito
- **Justification:** Using AWS Cognito simplifies the implementation of secure user authentication and access control, ensuring that only authorized users from the University of Calabria can access the system.

5. Web Dashboard

- **Design Solution:** Page Application using Amplify
- **Justification:** provide a seamless user experience by loading content dynamically without refreshing the entire page. AWS Amplify facilitates the creation and deployment of the webservice, ensuring an efficient and responsive web interface for users to monitor data and manage settings.

6. Email Notifications

- **Design Solution:** Node-Red using SMTP Gmail Server.

7. Voice Alerts

- **Design Solution:** MQTT Communication to Secondary ESP32
- **Justification:** Using MQTT for communication between the primary and secondary ESP32 devices ensures low latency and reliable message delivery. This is crucial for triggering voice alarms promptly when thresholds are exceeded.

Table 1: Design table

Requirement	Design Solution
Real-time Data Monitoring	MQTT Protocol (mosquitto)
Data Storage	NoSQL Database (Amazon DynamoDB)
Threshold Alerts	Event-driven Architecture
User Authentication	AWS Cognito
Web Dashboard	Amplify and CodeCommit
Email Notifications	Node-Red
Voice Alerts	MQTT Communication to Secondary ESP32

Implementation

the technologies that have been used are AWS Services represented by Cognito and Lambda for Authentication and Authorization, DynamoDB as NoSQL DB to store the sensed values, Amplify to deploy the web framework from the CodeCommit repository which cloned the data from S3 storage, then using EC2 to host the an Ubuntu machine that contain Node-Red and Mosquitto as a broker to MQTT protocol and finally managing the policies and roles using IAM. *Two ESP32 for sensing using DHT11 and Alarming using lights and Buzzer.*

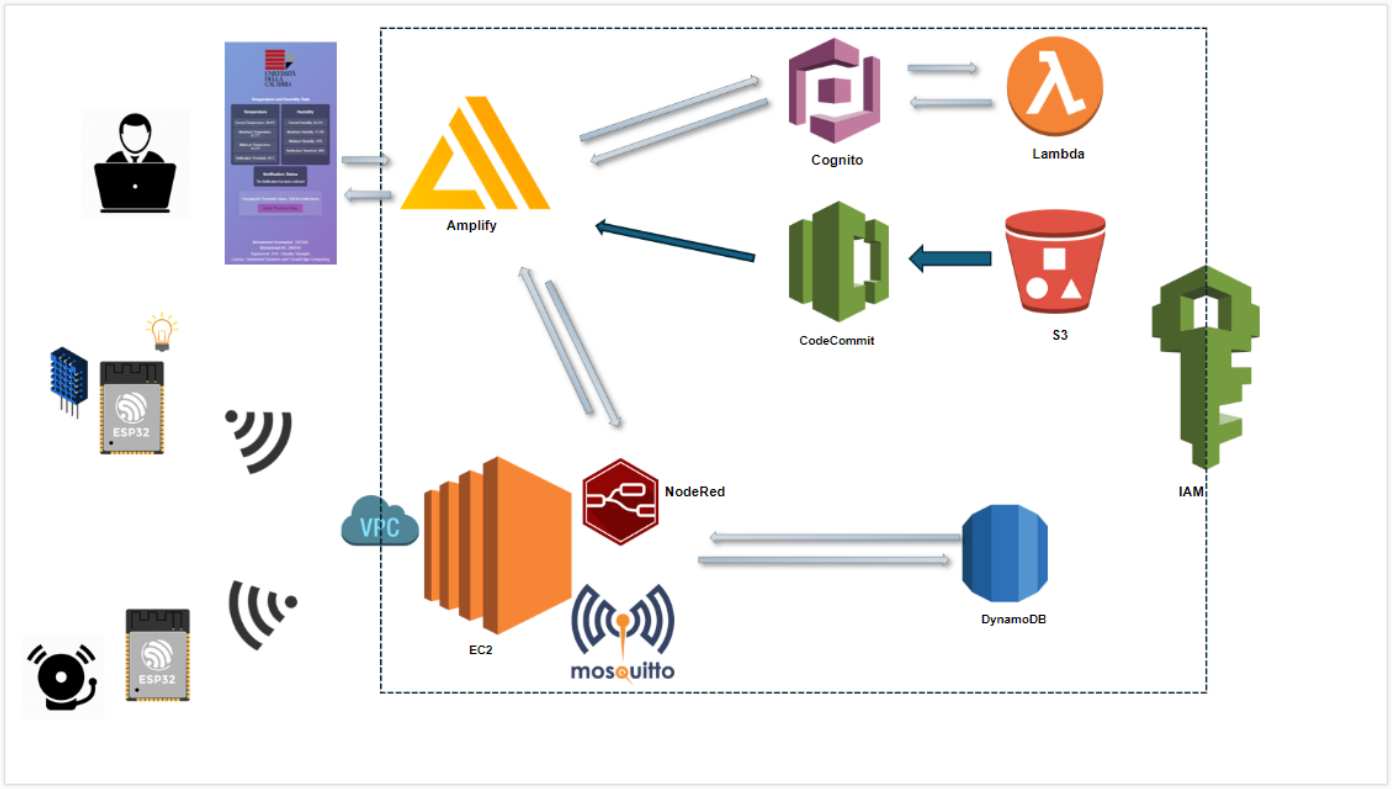

Figure 2: Deployment

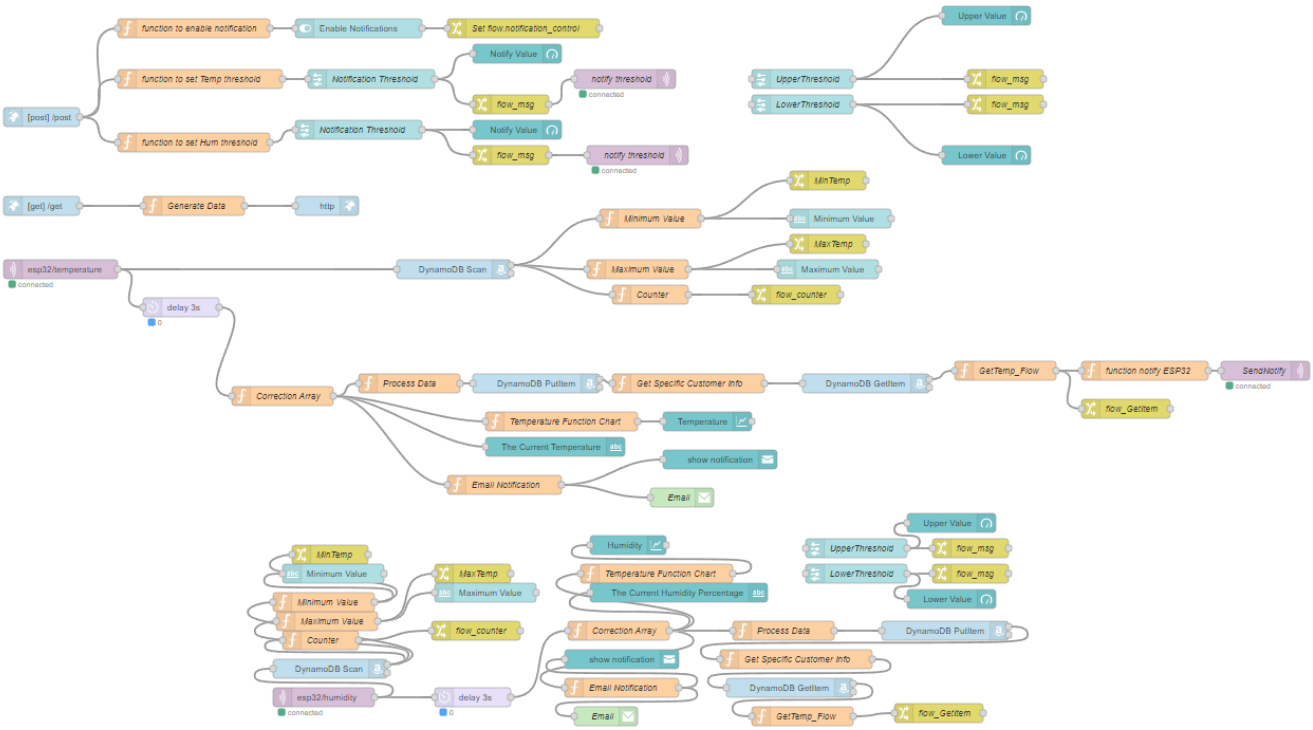

Figure 3: Node-Red flows

Node-Red flows:

Starting by receiving the data by subscribing to the proper topics using MQTT nodes, then scan the table in case have previous values to ID them by numbers, and to search for the maximum and minimum values in the table using function nodes, then send their values and the values od the total number od items to the flow nodes.

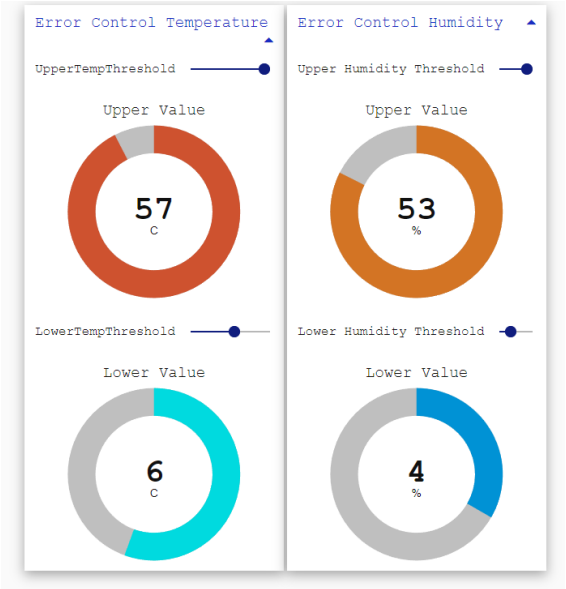

Figure 4: Correction area

Meanwhile the new values will be delayed by 3 seconds (to ensure that the procedure of the scanning has been done the counter has been sent to the flow) because the time of sensing the data and send them to the broker is each 5 seconds, the new values will enter a correction array function node to ensure that the sensed data is in a correct range (due to not perfect design of DHT11 it may send unreasonable data), and this range is controlled by the dashboard of the node-red as in figure 4, using dashboard nodes and flow change nodes.

Then into two nodes the firs email function node to transmit after checking the value and the threshold and the notification enabled or not to an email node, to send an email to the user state the value and notify, the other node is processing function node to make the payload of previous nodes suitable to be sent to the table in the Dynamodb cloud by using AWS node and DynamoDB as a service and the method is PutItem. In a purpose od checking the delay that could happen the system will retrieve the last value and check it if it above the threshold it will send to the other ESP node to set the alarm on or off, and also send it to the flow.

Now the http protocol have been introduced with http in and out nodes for the POST and GET methods by using the port 65500 TCP and Certificates to secure the communication with the web. By using the POST method a data that has been sent by the web to the node-red to modify the needed values.

The data will be displayed on the Node-Red Dashboard as it is clear in the figure 5 below, with the availability to control the thresholds and the activation state of the notification.

Web Application:

The web application is deployed using Amplify AWS, and the CodeCommit AWS for maintaining and development, two main pages one for monitoring and the other for controlling, the control pages to updates the values of thresholds and the activation of the notification, but after sign in or Register if the user has an email address from University of Calabria, served by Cognito and Lambda AWS. Figure 6 is the show of the web application on the browser.

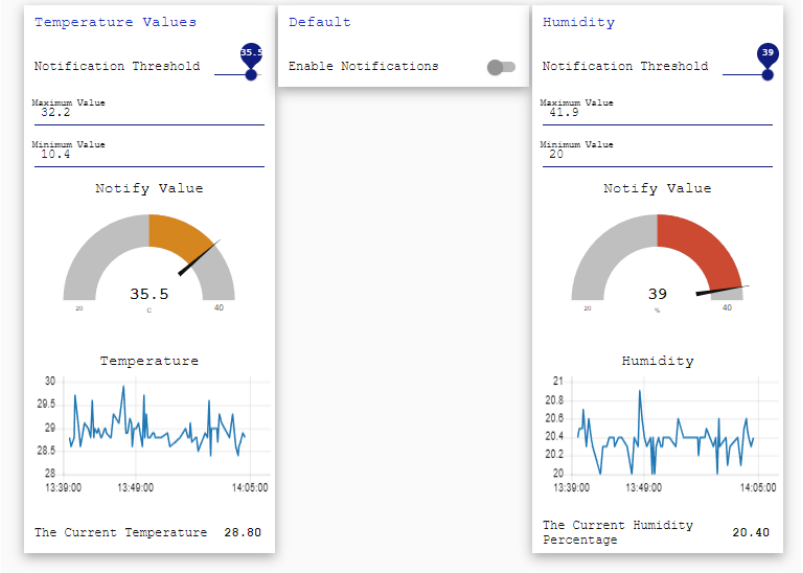

Figure 5: Node-Red dashboard monitoring and control

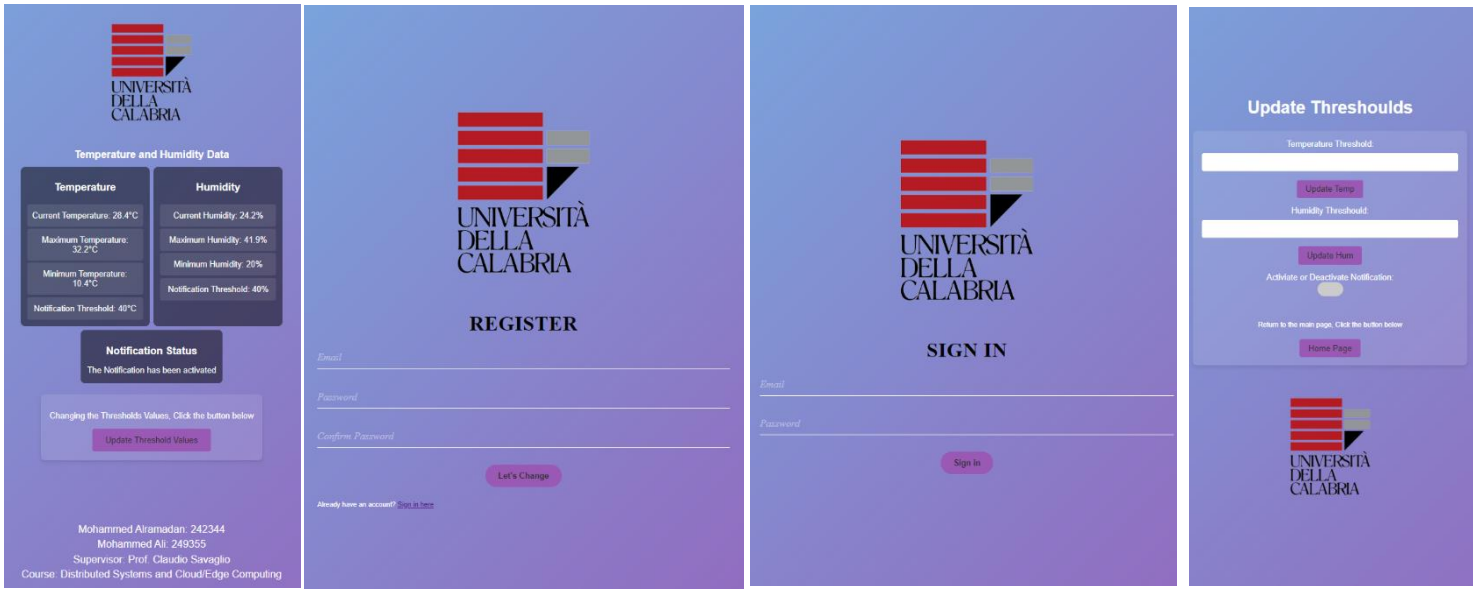

Figure 6: Web Service interfaces

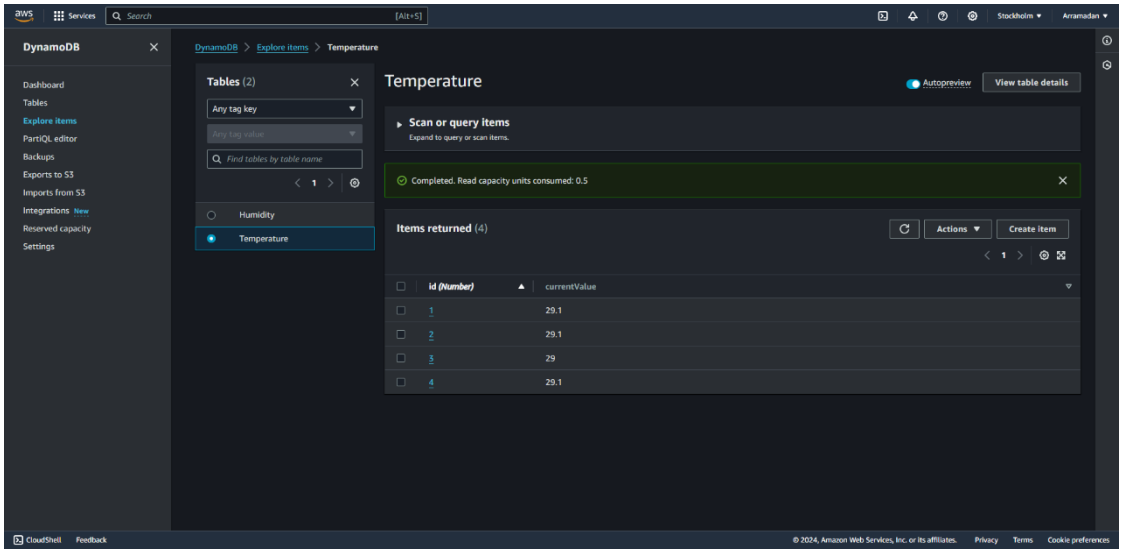

Figure 7 shows the DynamoDB AWS service and two tables for humidity and temperature with values of the temperature.

Figure 7 Tables in DynamoDB

ESP32

Two ESP32 has been used to sense and transmit the data, the other will notify the user in case of need. First one is using DHT11 as a sensor for humidity and temperature and a LED as a quick example to a replacement of Air Condition in case the Temperature has increased upon specific value and the threshold in the esp32 also controlled and it will be changed if the user has change it either from the web Application or from the Node-Red dashboard. Second ESP connected and subscribed to a topic called notify in the broker that is modified by the node-red processing. Figure 8 is the Arduino IDE that has been used to upload

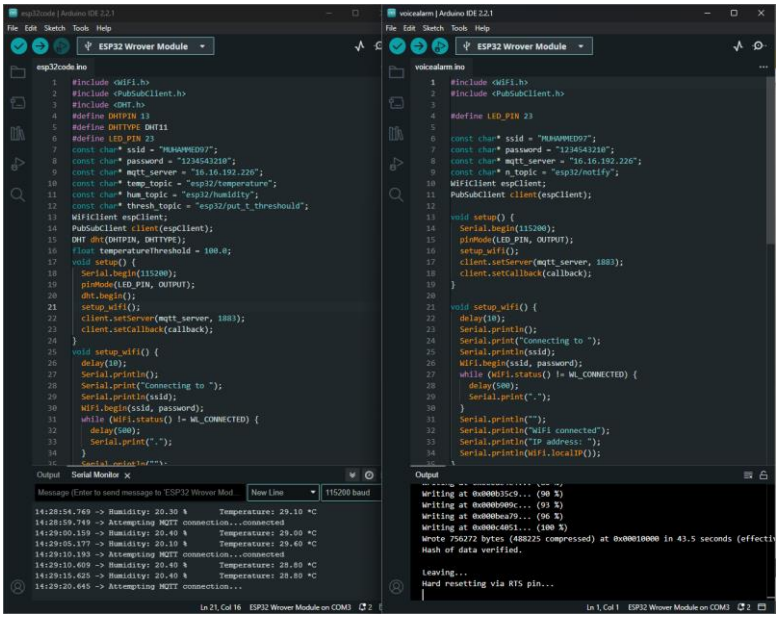

Figure 8: Arduino IDE forESP32 codes

the codes on the esp32 and to monitor for troubleshooting purposes using serial monitor.

Implementation Choices table

Table 2: Implementation choices

Requirement	Current Design Solution	Alternative Options	Justification for Current Solution
Real-time Data Monitoring	MQTT Protocol (Mosquitto)	HTTP, CoAP, AMQP	MQTT is lightweight, and designed for reliable real-time communication, making it more suitable for IoT applications compared to heavier protocols like HTTP or less efficient protocols like CoAP.
Data Storage	NoSQL Database (Amazon DynamoDB)	SQL Database, InfluxDB	NoSQL databases like DynamoDB are highly scalable and flexible, making them more suitable for handling time-series data compared to traditional SQL databases. InfluxDB is a strong alternative but less integrated with AWS.
Threshold Alerts	Event-driven Architecture (Node-RED)	Traditional Polling, Custom Scripts	Node-RED's event-driven architecture allows for real-time responses to data changes, reducing latency and processing overhead compared to traditional polling or custom script-based approaches.
User Authentication	AWS Cognito	Custom Authentication System, OAuth	AWS Cognito provides a robust, ready-made solution for secure authentication and integration with other AWS services, offering a quicker and more secure setup than building a custom system.
Web Dashboard	AWS Amplify and CodeCommit	Custom Frontend with React/Vue, GitHub Pages	AWS Amplify simplifies the creation and deployment process, offering built-in support for backend integration, CI/CD pipelines, and hosting, which reduces development and maintenance efforts compared to custom setups.
Email Notifications	Node-RED using SMTP Gmail Server	AWS SES, Custom SMTP Server	Node-RED provides an easy-to-configure and flexible environment for handling email notifications, whereas AWS SES and custom SMTP servers might require more setup and maintenance effort.
Voice Alerts	MQTT Communication to Secondary ESP32	HTTP, WebSocket, Zigbee	MQTT ensures low latency and reliable delivery specifically designed for IoT applications, making it superior to HTTP and WebSocket in this context. Zigbee could be an alternative but requires additional hardware.
Web Application Communication	HTTP Protocol for POST and GET requests	WebSocket, gRPC	HTTP is a standard, widely used protocol for web communication, offering compatibility and simplicity. WebSocket and gRPC provide real-time capabilities but are more complex to implement and maintain for basic GET/POST needs.

Conclusion

This project successfully satisfied most of the outlined requirements, delivering a robust environmental monitoring system using IoT and AWS Cloud Services.

Fully Addressed the Requirements**:** Real-time Data Monitoring**:** Achieved through the use of the MQTT protocol, ensuring continuous and efficient data transfer from ESP32 devices to the cloud server. Data Storage**:** Implemented using Amazon DynamoDB, which efficiently stores time-series data, providing quick read/write operations essential for real-time applications. Threshold Alerts**:** The event-driven architecture using Node-RED promptly triggers alerts when thresholds are exceeded, ensuring timely notifications. User Authentication**:** AWS Cognito and Lambda was effectively used to secure user authentication, limiting access to authorized users from the University of Calabria. Web Dashboard**:** Deployed using AWS Amplify and codecommit, offering a seamless and responsive user interface for data monitoring and management. Email Notifications**:** Configured using Node-RED and an SMTP Gmail server, providing reliable email alerts for threshold breaches. Voice Alerts**:** Utilized MQTT communication between the primary and secondary ESP32 devices, ensuring prompt voice alarms.

Partially Addressed Requirements**:** Security**:** While data access is restricted to authenticated users, future enhancements could include more granular access controls and advanced encryption techniques to further secure data transmission and storage. Reliability**:** Although sensor errors were minimized, integrating more advanced error correction algorithms and additional redundancy in data collection could further enhance system reliability.

Future Developments: Smart Irrigation**:** Future iterations could integrate a more sophisticated irrigation model, taking into account factors such as plant age, species characteristics, and evapotranspiration rates. This would make the irrigation process more efficient and tailored to specific plant needs. Enhanced Data Analytics**:** Introducing machine learning models to analyze historical data for predictive insights could enhance decision-making and optimize environmental conditions in the greenhouse. Extended Sensor Network**:** Expanding the sensor network to include additional parameters such as soil moisture, light intensity, and CO2 levels could provide a more comprehensive monitoring solution.

In conclusion, this project demonstrated the effective use of IoT and AWS Cloud Services for environmental monitoring (Greenhouse), providing real-time data and efficient solutions for smart greenhouse applications. With future developments, the system can be further optimized to enhance agricultural productivity and environmental control.