UNIVERSITY OF CANBERRA

LAB REPORT

Unit No	**11511**
Unit Title	**Internet of Things**
Group Number	**Group 4**

Student Number	**Student Name**
u3198718	Kit Chaivannacoopt
u3261669	Maxwell Dauda
u3253332	Nafis Khan
u3241483	Sonam Chophel

Abstract

The Smart Home Automation System project aims to design and implement an IoT-based intelligent monitoring and control solution for residential environments. Using the KS5009 Smart Home Kit, ESP32 microcontroller, and Raspberry Pi 4000, the system integrates multiple environmental sensors and appliance controllers. Sensor data such as temperature, humidity, motion, and appliance usage events are collected, processed locally, and forwarded to AWS IoT Core for remote accessibility. A local InfluxDB database captures real-time and historical data, while Grafana dashboards visualise environmental trends and appliance activities. Python scripts enable data routing and cloud connectivity, ensuring secure and scalable operation through MQTT protocols. The solution supports automated appliance control based on sensor feedback, improving energy efficiency and user convenience. The project demonstrates a practical and scalable approach to IoT-enabled home automation, cloud data management, and real-time visualisation.

Introduction

This project focuses on the development of a comprehensive Smart Home Automation System, combining edge computing on a Raspberry Pi, microcontroller-based sensing via an ESP32, and cloud-based data management through AWS IoT Core.

The system employs a variety of sensors, including a DHT11 temperature and humidity sensor, PIR motion detectors, gas sensors, and rain sensors, to monitor the household environment. It also integrates actuators, such as relay-controlled lights and fans, and servo motors for automated door and window operations. Sensor readings and appliance events are transmitted using the MQTT protocol to a local Mosquitto broker running on a Raspberry Pi. A Python-based bridge script processes incoming data, forwarding it securely to AWS IoT Core and writing structured entries into an InfluxDB time-series database for historical tracking.

Visualisation and analytics are provided via Grafana dashboards, which present real-time and historical environmental trends, appliance usage metrics, and occupancy patterns. This architecture supports remote access, automated decision-making based on sensor inputs, and detailed insights into household behavior. The project not only demonstrates the technical integration of IoT components but also addresses broader considerations such as sustainability, energy conservation, and the future potential of smart living environments.

System Architecture

Hardware Components

The Smart Home Automation System utilises the KS5009 Smart Home Kit alongside additional networking and processing hardware. The major hardware components include:

ESP32 Microcontroller: Captures environmental sensor data (temperature, humidity, gas, rain) and appliance events (relay control, servo control). Publishes sensor and event data over Wi-Fi using the MQTT protocol.

Raspberry Pi 4B: Acts as the IoT edge hub. Hosts a local MQTT broker (Mosquitto) to receive data from the ESP32. Executes Python scripts to process incoming messages, store data locally, and forward relevant data to AWS IoT Core.

Sensors:

- **DHT11 Sensor:** Measures temperature and humidity.
- **PIR Motion Sensor:** Detects occupancy within a room.
- **Gas Sensor:** Monitors gas concentration for safety alerts.
- **Rain Sensor:** Detects external rainfall.

Software Actuators:

- **Relay Modules:** Enable control of household appliances (lights, fans).
- **Servo Motors:** Automate door and window movements based on control signals.
- **LEDs and Buzzers:** Provide immediate visual and audible feedback for certain events (e.g., motion detected, hazard alert)

Software Components

Arduino IDE: Used for developing and uploading firmware to the ESP32, enabling sensor reading, actuator control, and MQTT publishing.

Mosquitto MQTT Broker (Local on Raspberry Pi): Manages lightweight messaging between ESP32 and Raspberry Pi.

Python Bridge Script: Subscribes to local MQTT topics, writes sensor and event data into InfluxDB and publishes all data to AWS IoT Core using secured TLS connections.

AWS IoT Core: Acts as the cloud-based MQTT broker and enables remote monitoring and data forwarding to cloud services.

InfluxDB: Local time-series database on Raspberry Pi for storing environmental and appliance event data.

Grafana: Visualises real-time and historical data from InfluxDB and Provides dashboards for environmental trends and appliance usage patterns.

Communication Flow

- The ESP32 collects data from attached sensors and publishes:
- Environmental readings to esp32/data
- Appliance events to esp32/events
- The Raspberry Pi, running a local MQTT broker, receives the data and:
- Stores it into InfluxDB for historical tracking.
- Forwards it securely to AWS IoT Core for cloud monitoring.
- Grafana pulls from InfluxDB to visualise sensor and appliance data.

Implementation

Hardware Setup

The initial phase involved assembling the hardware components of the Smart Home Automation System. The KS5009 Smart Home Kit provided the necessary sensors (DHT11, PIR motion sensors, gas sensor, rain sensor) and actuators (relays, servo motors, RGB LEDs, and buzzers).

The ESP32 microcontroller was used to interface with all input and output devices. Sensors were connected to appropriate GPIO pins, ensuring proper pull-up or pull-down resistor configurations where needed (e.g., PIR motion sensor). Actuators such as relays for appliances (lights, fans) and servo motors for window and door automation were also connected to the ESP32, utilising PWM signals where appropriate.

The Raspberry Pi 4B was configured as the central IoT hub, operating on a local Wi-Fi network. Mosquitto, a lightweight MQTT broker, was installed to handle the communication between the ESP32 and the Raspberry Pi. Power supply and network configurations were tested to ensure reliable operation of the system.

ESP32 Programming

<u>The ESP32 was programmed using the Arduino IDE. Key libraries included:</u>

- WiFi.h for Wi-Fi connectivity.
- PubSubClient.h for MQTT communication.
- DHT.h for reading temperature and humidity data.
- Servo.h for controlling servo motors.
- Adafruit_NeoPixel.h for RGB LED control.

<u>The ESP32 was programmed to:</u>

- Read sensor values at regular intervals (e.g., every 5 seconds).
- Control actuators based on HTTP requests received (e.g., turning lights ON/OFF opening/closing windows and doors).
- Publish sensor data to the MQTT topic esp32/data in JSON format.
- Publish appliance events (e.g., “LED ON”, “Fan OFF”) to the MQTT topic esp32/events in JSON format whenever an actuator was triggered.
- The ESP32 maintained a persistent Wi-Fi connection and reconnected to the MQTT broker automatically if the connection dropped.

Raspberry Pi Configuration

<u>The Raspberry Pi was set up with the following key components:</u>

- Mosquitto Broker: Installed via package manager (apt) and configured to listen for incoming MQTT messages from ESP32.
- Python 3 Bridge Script: Developed to handle multiple MQTT subscriptions:
- esp32/data for environmental sensor readings.
- esp32/events for appliance control events.

The Python script performed several critical tasks:

- Subscribing to both MQTT topics.
- Processing sensor data by writing it into an InfluxDB database under the environment_data measurement.
- Processing appliance event data separately into the appliance_events measurement.
- Forwarding all received data securely to AWS IoT Core using TLS certificates for encrypted cloud communication.

The script ensured efficient local data storage while enabling real-time cloud updates.

AWS IoT Core Integration

AWS IoT Core was configured to enable secure communication between the Raspberry Pi and the cloud. Key steps included:

- Creating an IoT Thing (device identity).
- Generating and attaching X.509 certificates for authentication.
- Defining a custom IoT policy allowing publish, subscribe, and receive actions on all topics.
- Setting up the MQTT Test Client to verify incoming messages from the Raspberry Pi.

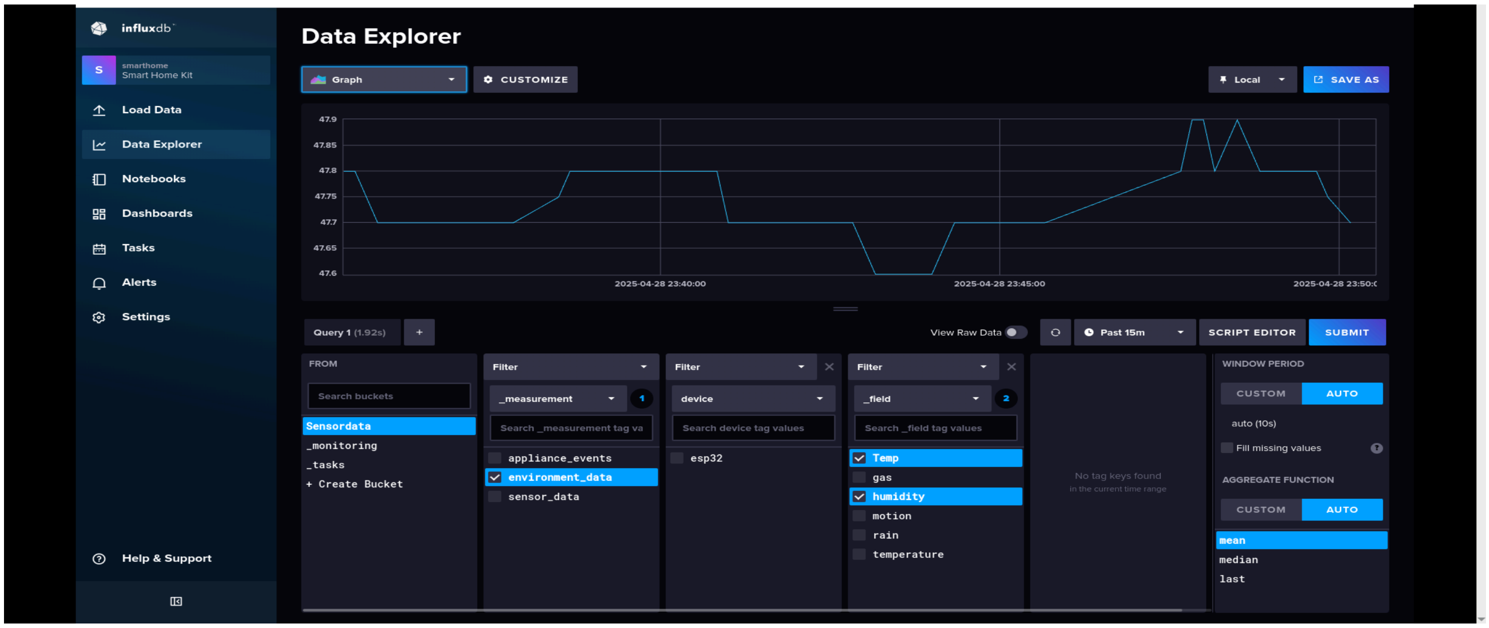
The Raspberry Pi bridge script utilised the AWS IoT Device SDK to establish a secure MQTT connection and publish sensor and event data to AWS, making it accessible for remote monitoring.

InfluxDB and Grafana Setup

InfluxDB, a time-series database, was installed on the Raspberry Pi to store historical data locally:

- Environmental readings were stored under the environment_data measurement.
- Appliance event statuses were stored under the appliance_events measurement.
- Grafana was installed and configured to visualise this data:
- A time-series panel was created to display temperature, humidity, gas, rain, and motion trends.
- A table panel was set up to display recent appliance activation/deactivation events.
- Thresholds and color rules were added to enhance visualisation (e.g., red for “OFF”, green for “ON”).

Grafana dashboards were configured to auto-refresh every 5-10 seconds, ensuring near real-time monitoring of household conditions.



Results

System Functionality Overview

The Smart Home Automation System successfully demonstrated the integration of sensors, actuators, and cloud connectivity through the IoT architecture design. The ESP32 microcontroller reliably captured real-time environmental data and appliance events, publishing structured JSON messages to local MQTT topics. The Raspberry Pi, acting as a local edge hub, successfully processed incoming data, stored it into the InfluxDB database, and securely forwarded it to AWS IoT Core.

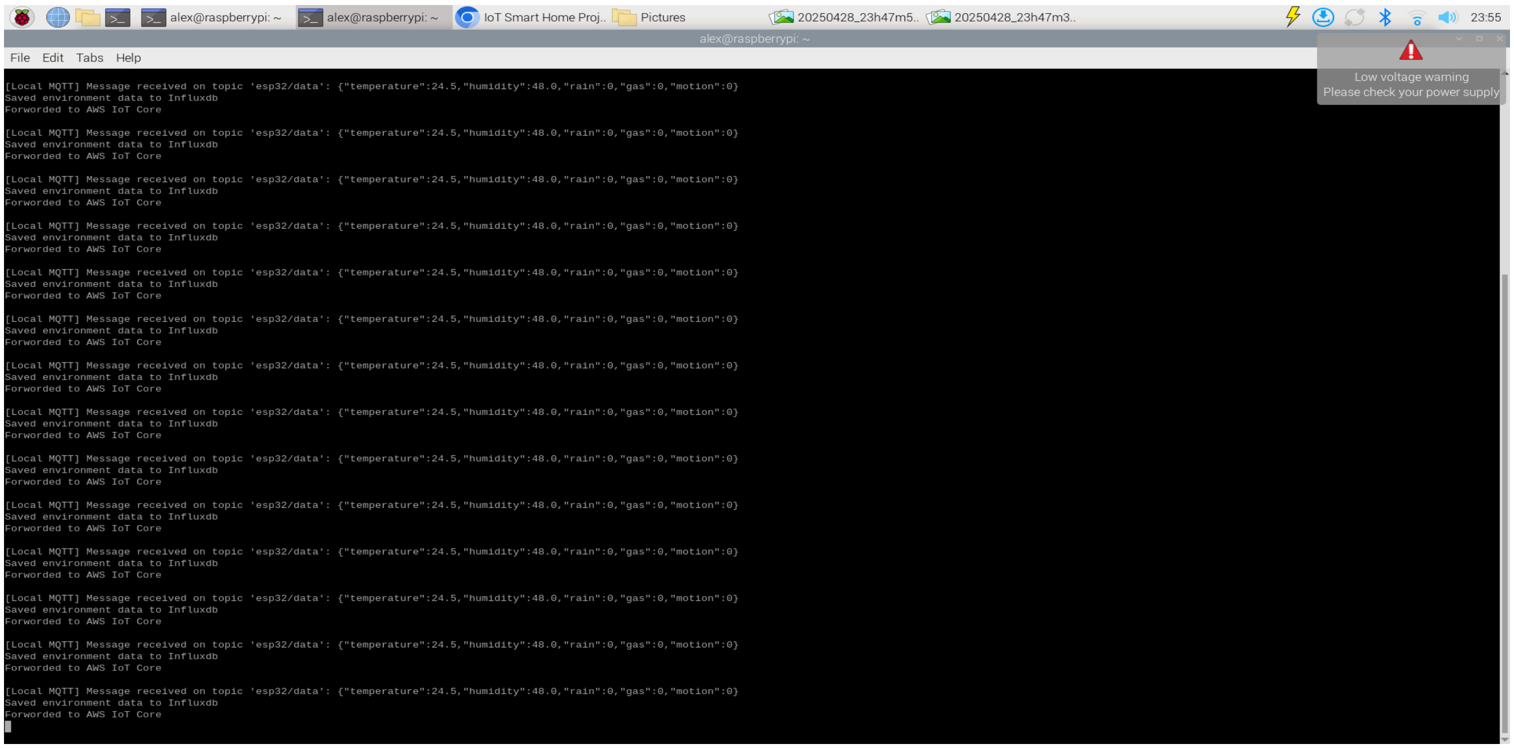
Local Monitoring and Database Logging

Upon system activation:

Environmental Sensor Data: Temperature, humidity, rain detection, gas readings, and motion detection values were received at the local MQTT broker (esp32/data) and logged into the environment_data measurement in InfluxDB.

Appliance Control Events: Each activation and deactivation of relays, LEDs, or servo motors triggered event messages published to the esp32/events topic, stored into the appliance_events measurement.

Raspberry Pi Console Output:



```
[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core

[Local MQTT] Message received on topic 'esp32/data': {"temperature":24.5,"humidity":48.0,"rain":0,"gas":0,"motion":0}
Saved environment data to Influxdb
Forworded to AWS IoT Core
```

Captured live incoming MQTT messages, database write confirmations, and AWS IoT Core forwarding logs.

Examples included:

MQTT Published: { "temperature": 25.4, "humidity": 48.3, "rain": 350, "gas": 200, "motion": 1 }

Saved appliance event to InfluxDB: Fan ON

AWS IoT Core Cloud Monitoring

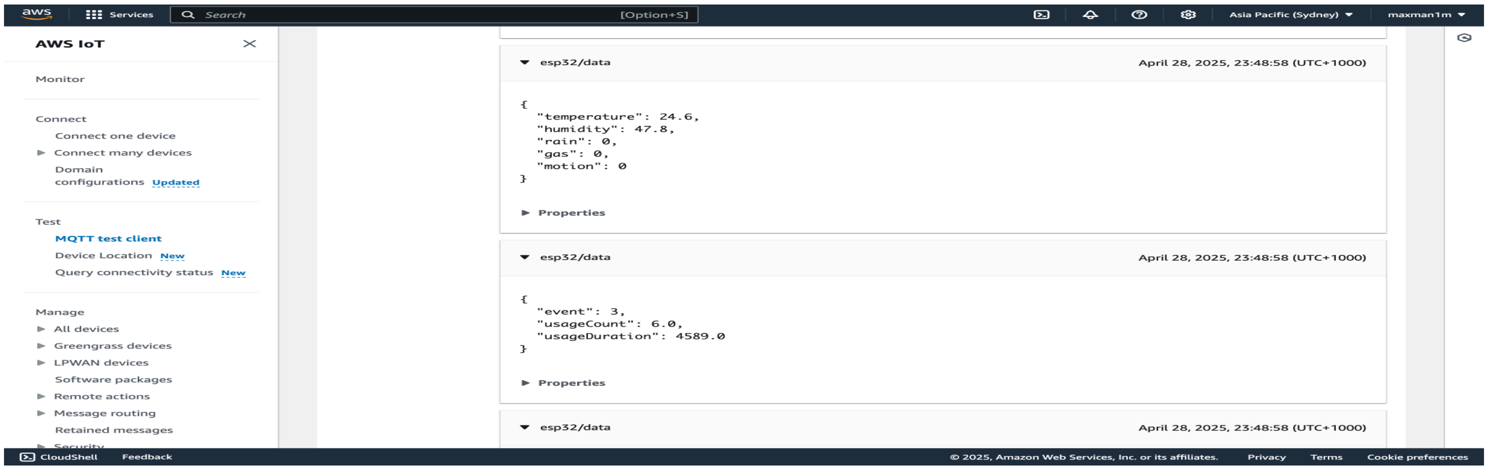
In AWS IoT Core's MQTT Test Client:

Real-time sensor readings and appliance event messages appeared under the expected topic structure (esp32/data, esp32/events).

Payloads were parsed correctly, reflecting live changes as triggered on the ESP32 hardware.

AWS IoT Test Client Screenshots:

- Subscribed topics successfully received updates every ~5 seconds.
- JSON messages matched the ESP32 published structure.



esp32/data April 28, 2025, 23:48:58 (UTC+1000)

```
{
  "temperature": 24.6,
  "humidity": 47.8,
  "rain": 0,
  "gas": 0,
  "motion": 0
}
```

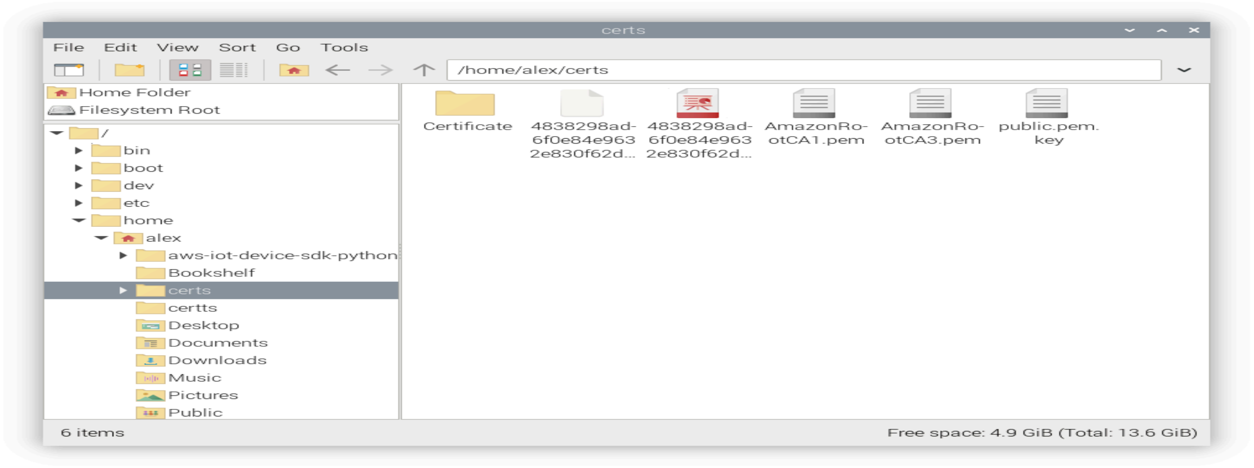
Properties

esp32/data April 28, 2025, 23:48:58 (UTC+1000)

```
{
  "event": 3,
  "usageCount": 6.0,
  "usageDuration": 4589.0
}
```

Properties

esp32/data April 28, 2025, 23:48:58 (UTC+1000)



(Public Key, Device Key, Private Key, Certificates)

Visualisation through Grafana Dashboards

Grafana dashboards were configured and produced live real-time monitoring of the system:

Environmental Trends:

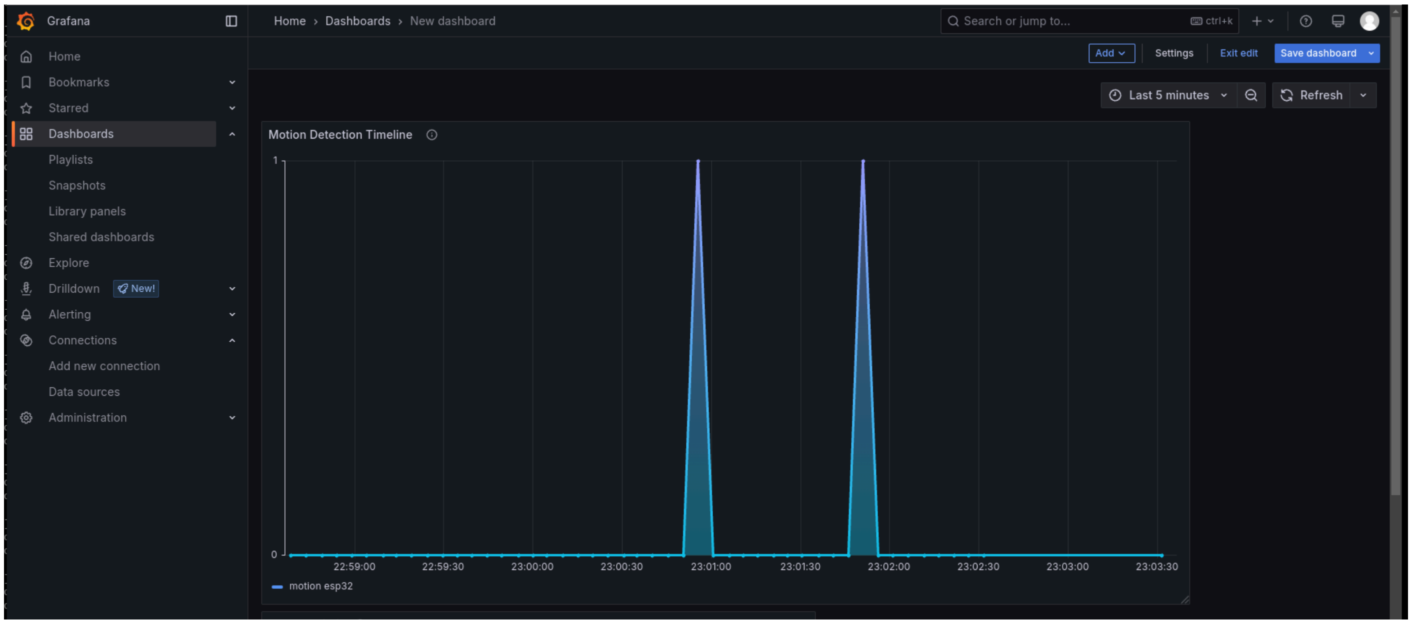
- Line graphs displayed variations in temperature and humidity over time.
- Gas sensor values and rain detection values were plotted for hazard monitoring.

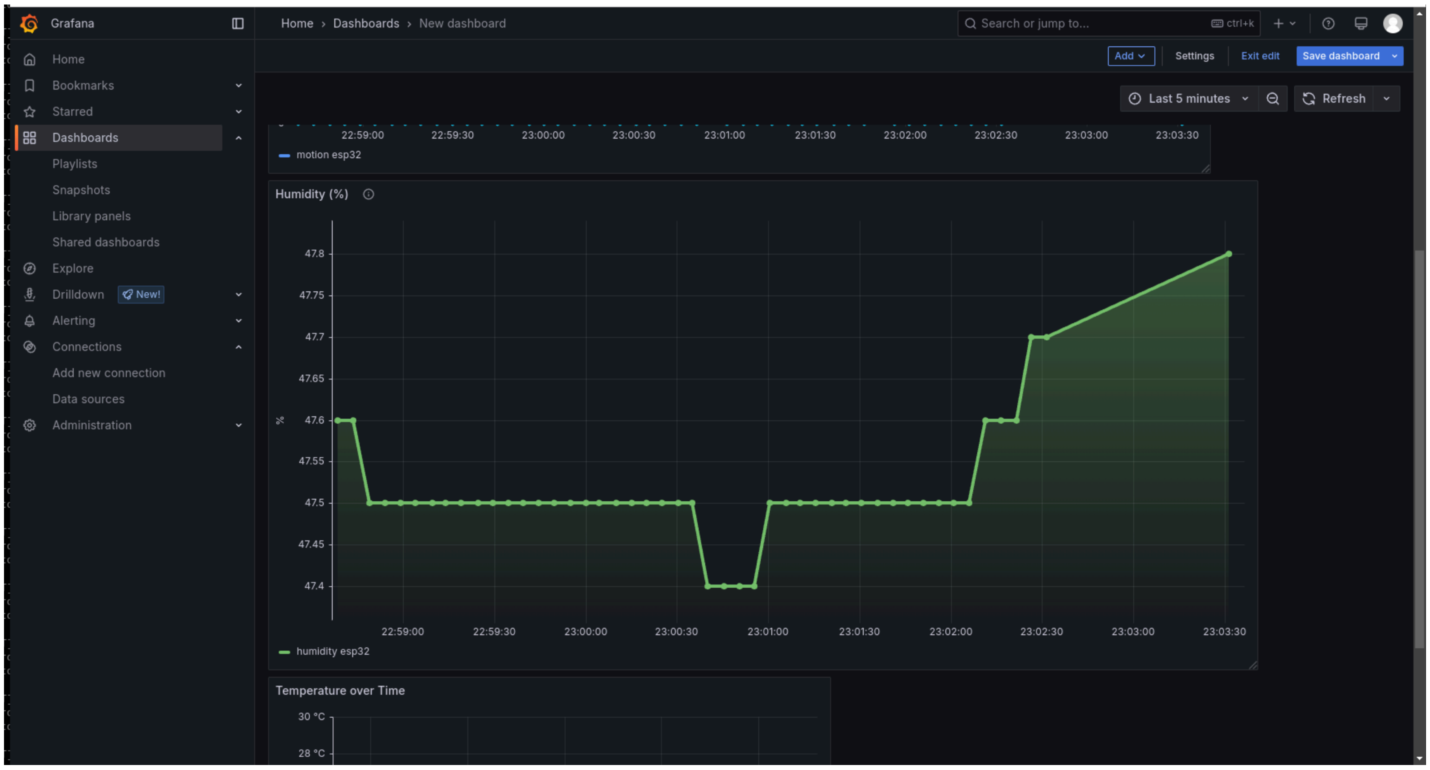
Motion detection events appeared as state changes (e.g., 0 to 1).

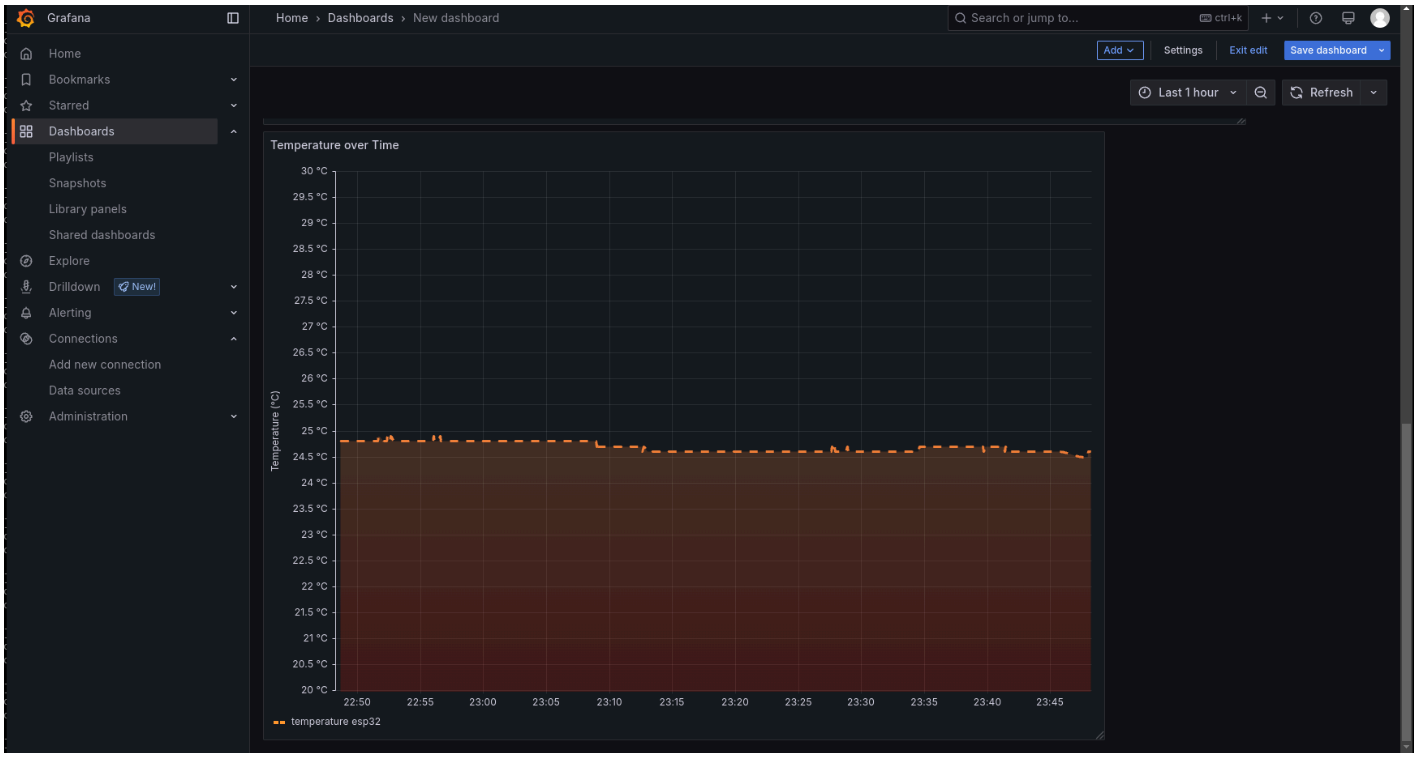
Appliance Usage:

- Appliance ON/OFF events were captured in a table visualisation.
- State timelines illustrated activation patterns across a day.

Grafana Dashboard Screenshots:







Challenges Faced

During the development of the Smart Home Automation System, several technical and operational challenges were encountered. Addressing these issues systematically contributed significantly to the successful completion of the project.

Telegraf Issues

The primary challenge encountered was in the data ingestion stage using Telegraf. We attempted to configure Telegraf's `mqtt_consumer` plugin to subscribe to the sensor data topics and write directly to InfluxDB. Despite following recommended configurations, the data was not being inserted into the database as expected. The Telegraf agent did not show obvious errors, but verifying the InfluxDB contents showed no new measurements. After spending considerable time on troubleshooting, we decided to switch to a simpler, more transparent method: a custom Python script. Once the Python script was in place, it successfully subscribed to MQTT and inserted data into InfluxDB, confirming that the pipeline was working.

MQTT Broker Connection Failures

Initially, the ESP32 was unable to establish a stable connection to the local Mosquitto MQTT broker running on the Raspberry Pi. This was due to the broker service not starting automatically after system boots. The issue was resolved by manually starting the Mosquitto service and enabling it to run on boot, ensuring consistent broker availability.

Multiple MQTT Topics Management

The need to handle both environmental data and appliance events required managing multiple MQTT topics (esp32/data and esp32/events). Early versions of the Python bridge script only subscribed to a single topic, causing missed messages. The script was later modified to subscribe to both topics simultaneously and process incoming messages based on the topic and payload content.

Data Type Handling in MQTT Payloads

A major challenge arose when processing different data types from MQTT messages. Environmental data (temperature, humidity, gas readings) were numerical, whereas appliance events were text strings (e.g., “LED ON”, “Fan OFF”). Attempts to treat all payload values as floats led to script crashes. This was corrected by implementing topic-based parsing, distinguishing between float and string values appropriately.

AWS IoT Core Certificate Management

Establishing a secure TLS connection to AWS IoT Core posed initial difficulties. Incorrect file paths for device certificates and private keys caused repeated authentication failures. After carefully verifying file locations and updating the Python script with the correct paths, secure communication with AWS IoT Core was established.

Visualisation Challenges in Grafana

Visualising both environmental data (continuous numeric trends) and appliance events (discrete on/off actions) within the same dashboard required careful design. Multiple panels were created:

- Time-series graphs for environmental monitoring.
- State timelines and tables for appliance event tracking.

This ensured clarity and usability in the final Grafana dashboard.

Network Stability and Latency

Occasional Wi-Fi instability during early testing led to short-term ESP32 disconnections from the MQTT broker. To mitigate this, the ESP32 firmware included automatic reconnection routines, while the Raspberry Pi bridge script incorporated retry mechanisms to maintain robustness.

Localising Data Storage

In the event of possible cloud failure to the AWS infrastructure and to improve overall availability of the Smart Home Data, implementing a system to locally collect the data and store it even after a failure might occur would prove vital for the system resilience. This challenge was handled within the Raspberry Pi integration as all the data reading is being forwarded locally to the time-series database (InfluxDB) within the Pi. The result of this design element ensured that the system would continue recording and storing data without the chance of loss if there is an issue with the AWS IoT Core.

Version Control

We also faced challenges with the software versions compatibility.The version of InfluxDB installed on the Raspberry Pi was not fully compatible with the version of Grafana we were using. This caused a connection error while trying to add Influxdb as the data source in Grafana. After troubleshooting we uninstalled the Influxdb and downloaded the Influxdb version which was compatible with grafana we were using. After the correct version was installed , Influxbd was connected to grafana successfully.

Conclusion

The Smart Home Automation System project successfully demonstrated the integration of hardware, software, and cloud technologies to create a functional, scalable, and efficient residential automation solution. By leveraging IoT principles, the system monitored environmental parameters, controlled household appliances, and provided real-time insights through cloud-based services and local dashboards.

The ESP32 microcontroller reliably captured sensor data and appliance events, communicating with a locally hosted Mosquitto MQTT broker on the Raspberry Pi. The Python bridge script effectively managed data routing, ensuring secure storage in InfluxDB and cloud forwarding to AWS IoT Core. Visualisation through Grafana dashboards provided real-time monitoring and historical analysis of environmental conditions and appliance usage trends.

Key project goals, including automated control, environmental monitoring, remote accessibility, and data-driven insights, were achieved. Additionally, challenges related to network connectivity, multi-topic MQTT management, certificate handling, and visualisation complexities were systematically addressed.

Reference

[1] Keyestudio, "KS5009 Keyestudio Smart Home," *Keyestudio Wiki*, [Online]. Available: https://wiki.keyestudio.com/KS5009_Keyestudio_Smart_Home. [Accessed: 28-Apr-2025].