Smart Room Environment Control

Group Assignment (Practical)

Subject Code: SWE30011

Subject Name: IoT Programming

Team Number: 1

Year & Semester: Semester 1 (2025)

Tutorial Day and Time: Friday 12:30pm

Word Count: 2349

Team Members

Student ID	Name
103844366	Khoi Ly
104762184	Ved Jay Makhijani
105159143	Jeren Tang

Table of Contents

1. Introduction

In the current times , there are more jobs that require people to spend more of their time indoors than outdoors. While staying indoors can protect us from the harsh environments of mother nature, such as rain and extreme hot weather, it requires the people to manually adjust the temperature of the air conditioner, close the door if too loud and if the room is too dark they have to manually turn on the light. While these maybe simple tasks for people to accomplish, having an automated system to manage these tasks will provide people with a more convenient way of living. Our team has developed an IoT application that will help maintain a stable room environment by turning on a fan if the temperature is too hot outside, the light will turn on if outside is too dark and close the door if outside is too loud. Our System also uses discord messaging service to deliver cloud based notifications to our user , suggesting whether to go outside or stay inside. It also delivers notifications when brightness , sound level and temperature exceed there thresholds, it also sends a daily sensor report which shows how many times light, sound and temperature exceeded there threshold. You can change the time this report is being sent via code.

We propose a system which will have two Arduinos, two edge devices and one cloud application. One Arduino will be outside the room, which will only have sensors to detect how bright, loud and hot the outside environment is. This data will be sent to the outside edge device which will be sent to the cloud edge device and will be sent to the cloud application (Thingsboard) and to the inside edge device. This data will be sent to the inside Arduino which will only have actuators such as a servo motor which controls the door, a fan to cool down the room, a LED to light up the room and an OLED display to show the user a message of what the device is doing. The Thingsboard will be accessible to the user and will show the user the Melbourne weather via OpenWeatherAPI™, all the statuses of the sensors and actuators and a graph of the temperature from the outside Arduino. We also have 2 modes for our system AUTO or MANUAL, in auto mode everything is done via pre defined thresholds and data from the weather API and the manual mode enables the user to control all the actuators themselves.

2. Conceptual design

The following is a diagram of how we implemented the IoT application. We have utilised the 3 layered architecture taught in the lectures which include the cloud layer, edge layer and IoT devices layer [1]. Initially, we developed the IoT application so that the inside edge and outside edge devices will both send to the cloud layer (ThingsBoard). However, we struggled with the MQTT connection and seem to only be able to connect to one edge device between the edge layer and cloud layer. To solve this problem, we decided to add another edge device which specifically was made to get data from both inside and outside edge devices and send the data via

MQTT to thingsboard which then provides us with a GUI to view all the data and control the actuators inside the room.

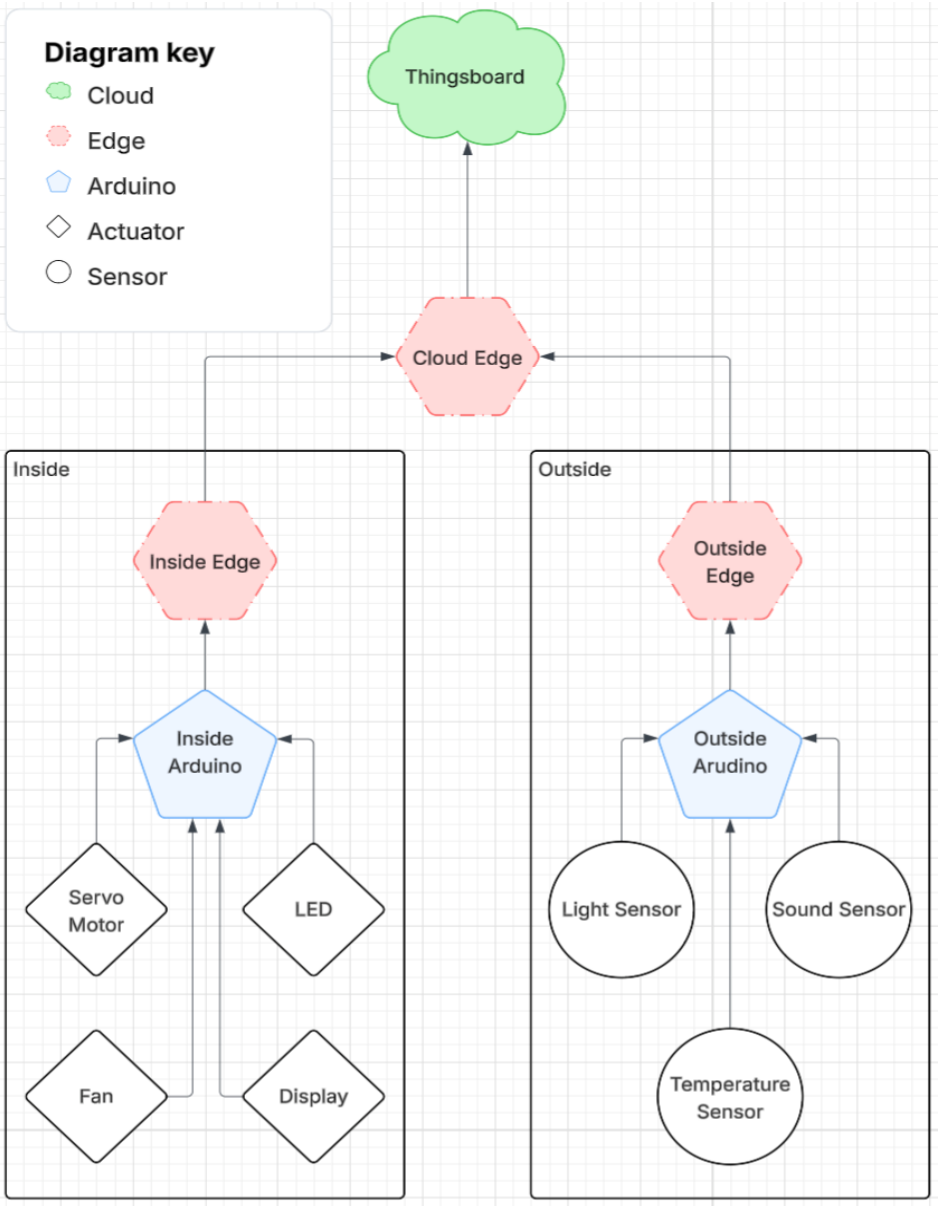

Figure 1. Systema architecture

3. Tasks breakdown

The following tab illustrates what each team member completed in this project. The tasks for this project were split into three parts which is based on the three-layered architecture style discussed in the lectures which included the cloud, edge and IoT things layer [1]. Khoi Ly did the edge layer and implemented the MQTT and Discord API in the edge devices. Ved did the cloud layer and created the UI in ThingsBoard and displayed buttons, sensor data, actuator data and weather API data. Jeren was responsible for implementing the IoT devices for the outside and inside devices and developing the sketches for both Arduinos.

Table 1. Responsibilities distribution.

Khoi Ly	Ved Jay Makhijani	Jeren Tang
1. Design system architecture. 2. Development of both the inside and outside edge servers' script. 3. Integrating all components to work together across three abstract layers. 4. Debugging. 5. Demonstrating of the system working via video recording. 6. Video editing.	1. Development of the cloud server script. 2. Designing and integrating ThingsBoard cloud GUI into the system. 3. Report writing. 4. Debugging cloud and cloud edge	1. Design system architecture. 2. Building the physical system. 3. Development of both the inside and outside Arduino scripts.

4. Implementation

The following section will discuss how our team implemented the design and explain how the design works.

4.1. IoT architecture

For this project we decided to utilise the 3 layered architecture style explain to us in the lectures [1]. This layered IoT architecture has three layers which include, Hardware Layer, edge layer and cloud layer. The Hardware layer is the layer where we made our hardware integrating all the necessary sensors such as light sensor, sound sensor , temperature sensor and our actuators such as led lights , led display with 2 Arduino uno R3's, These hardware systems send data to their respective edge servers via serial communication and the inside Arduino receives commands on actuator control via serial communication as well . In the edge layer, we implemented data collection, processing and sending logic to receive the data from the Arduinos via serial communication and database logic such as creating the database and storing data and sending the data to ThingsBoard via MQTT communication protocols. We also implemented MQTT protocols so that it can also receive data and commands from the cloud layer. We Implemented the cloud layer via a an IoT cloud Dashboard called ThingsBoard which gets the data from the cloud edge server Via MQTT and allows the user to view the data and interact with the actuators via GUI.

4.2. Sensing and actuation process

This project uses the following sensors:

1. Temperature Sensor
2. Light Sensor
3. Sound Sensor

The assembled system using the sensors, connected with the Arduino inside can be seen in Figure 2.

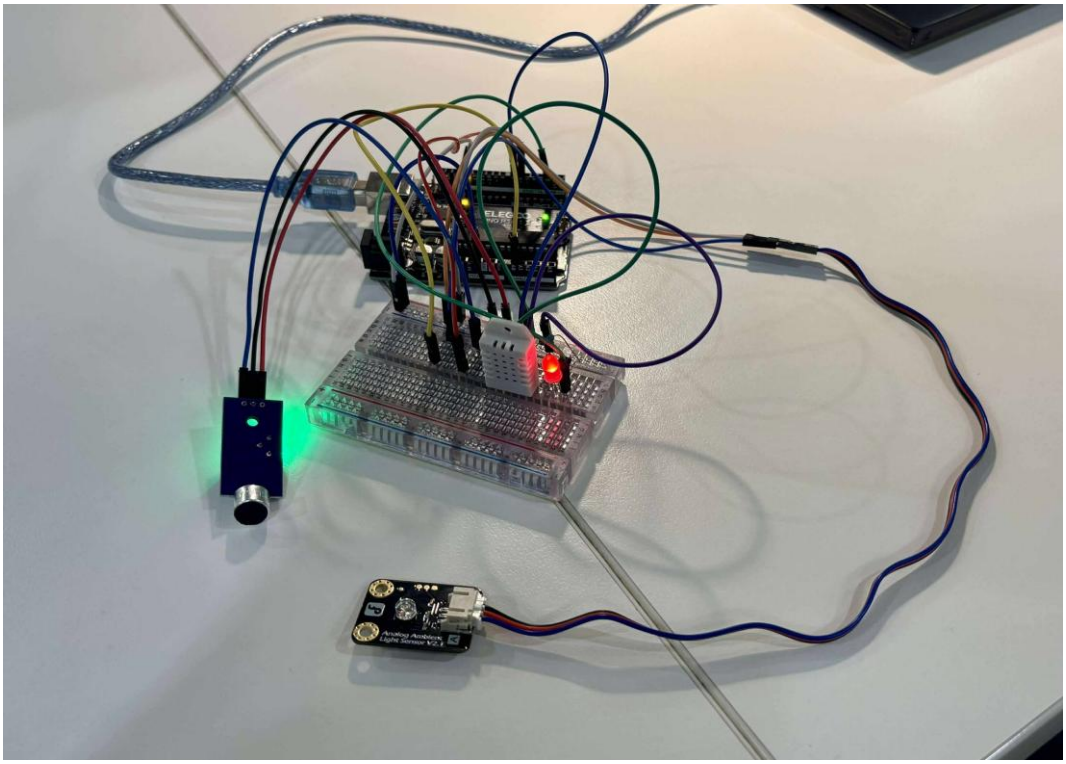

Figure 2. The sensors connected to the Arduino outside at the physical layer.

Similarly, the project uses the following sensors:

1. Display
2. Fan
3. Servo motor for door
4. LEDs

The assembled system with the actuators connected to the Arduino inside can be seen in Figure 3.

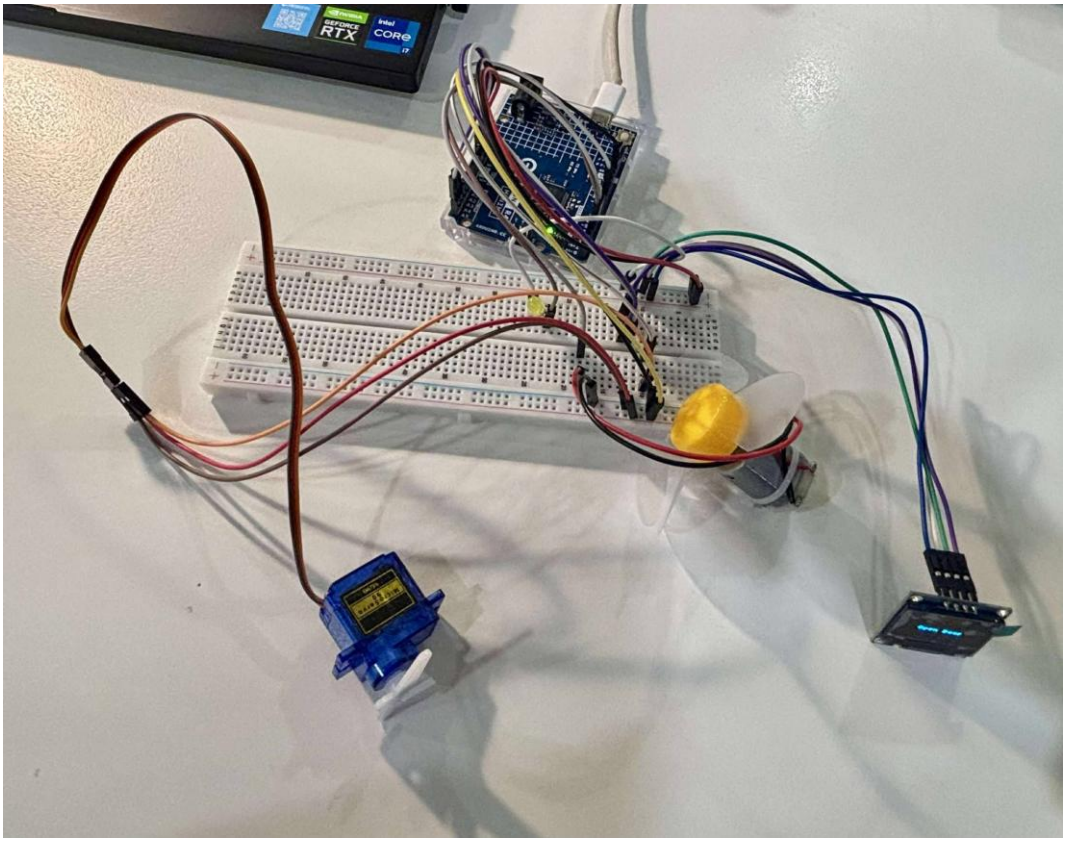

Figure 3. The actuators connected to the Arduino outside at the physical layer.

All the actuators are in the room. This is because, the inside Arduino is utilised to manage tasks such as notifying the user by showing a message through a display. For example, when the room is too hot the OLED display will show Cooling down and will open the door by using the servo motor and turn off the fan. The inside Arduino ensures that the room's environment is suitable for the user to able to focus on their work. The outside Arduino is designed to only have sensors which will send the data to the cloud and the inside edge which will then send it to the inside Arduino. With this data, the inside Arduino can manage if the actuators should be active or not. The table depicts the conditions and actuators configuration match the logic controlled by the Arduino can be found in Table 2.

Table 2. Sensors condition table controlling actuators state.

Conditions	Day, Hot, Loud	Day, Hot, Quiet	Day, Cold, Loud	Night, Hot, Loud	Day, Cold, Quiet	Night, Hot, Quiet	Night, Cold, Loud	Night, Cold, Quiet
LED	Off	Off	Off	On	Off	On	On	On
Fan	On	On	Off	On	Off	On	Off	Off
Door	Close	Close	Close	Close	Open	Open	Close	Open

4.3. Edge computing

In our implementation, there are three servers: inside edge server, outside edge server and cloud server. The inside and outside edge servers get the data from the sensors/actuators from the Arduino via serial communication. These 2 edge servers send the data via MQTT to the cloud server which serves different purpose. The cloud server acts a middleman to get data from the inside and outside Arduino and then merge the data together to send it to the ThingsBoard dashboard via MQTT communication. ThingsBoard cloud then utilizes this data to show it in the GUI and plot these values over a certain window of time.

4.4. Communication protocols

For this project we decided to use MQTT as the communication protocol between the edge layer and cloud layer. This is because it is a lightweight protocol and supported by the ThingsBoard cloud which allowed us to implement faster rather than using another protocol like CoAP that although lighter and faster does not guarantee deliver of message because it runs on UDP instead of TCP and lacks built in security features that MQTT has. We implemented MQTT by utilising the Cloud edge server as a manager of the MQTT protocol. This is because initially our team design the IoT device so that both edge devices from inside and outside will send data to the ThingsBoard cloud. However, we faced an issue with linking 2 devices to the ThingsBoard so we decided to solve this issue by implementing another edge device which will act as a middleman between the edge device for the Arduinos and the ThingsBoard. This way instead of sending two different messages to the Cloud we only need to send one. The cloud server is the publisher for the cloud and subscriber for the commands making it send data to ThingsBoard and receive commands for actuator control inside the room.

The MQTT topic structure was defined in Table 3. The transmitted data is under JSON format, making it easier for edge and cloud servers to parse and read.

Table 3. MQTT topics summary.

Topic	Functionality
edge/outside/data	Transmit sensors data
edge/inside/data	Transmit actuators state
cloud/control/led	Transmit led control command
cloud/control/fan	Transmit fan control command
cloud/control/door	Transmit door control command
cloud/control/mode	Transmit mode switch command
cloud/suggestion	Transmit suggestion message

4.5. API

In this project we decided to implement 2 types of APIs which include, Discord and OpenWeather API for Melbourne data. The Discord API is implemented in the “cloud” edge server and is used to send recommendations to the user to go outside if it is sunny or not. This condition is determined by the OpenWeather API and the ambient light sensor implemented in the outside Arduino. For this project we used the OpenWeather API to get data about the current conditions of Melbourne weather.

4.6. Cloud computing

The cloud computing platform is hosted by ThingsBoard and it allows the user to view the current state of the sensors such as how bright it is and the actuators like if the fan was on or off. It also allowed the user to control if the system is in manual or auto mode. They can also see the current temperature of the Melbourne weather and a graph showing the average temperature detected by the sensor from the outside Arduino. It also shows the temperature, light and sound outside of the room and allows user to control the door, fan and light via manual mode. The dashboard can be seen in Figure 4 and Figure 5.

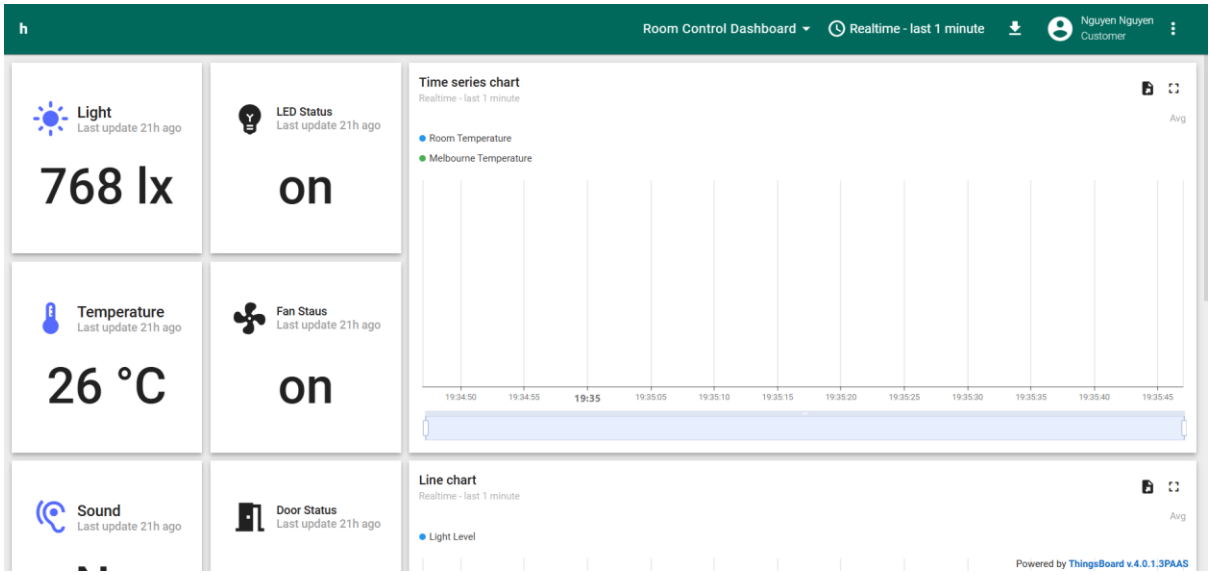

Figure 4. ThingsBoard dashboard.

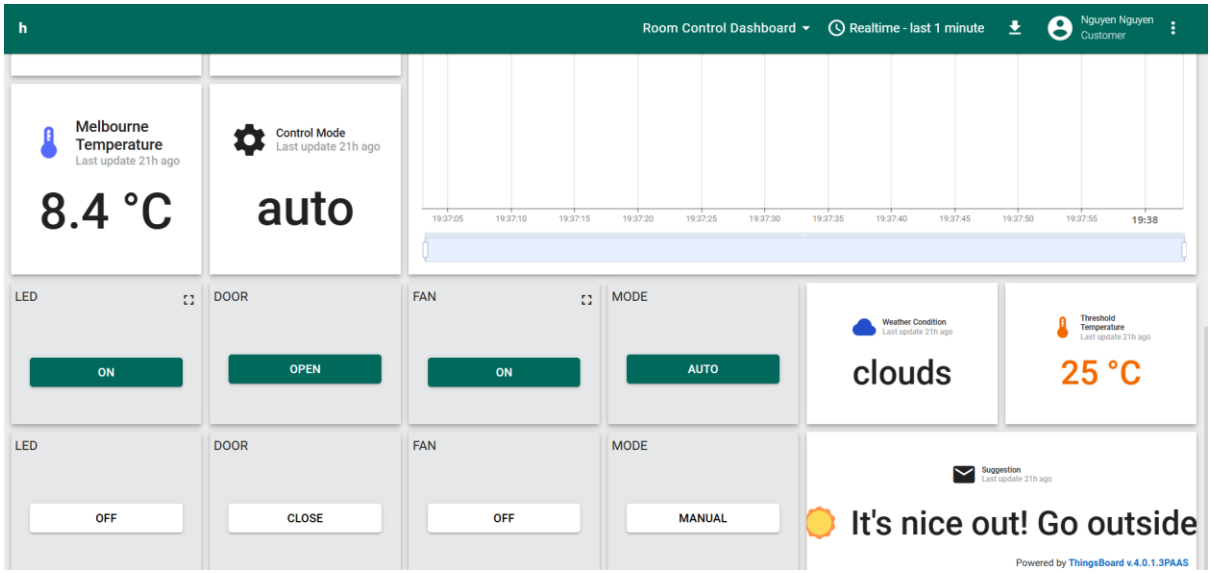

Figure 5.ThingsBoard dashboard (continue).

5. User Manual

5.1. Physical layer setup

1. Upload the Outside_Arduino sketch onto the Arduino outside.
2. Using the following table, connect the sensors and LED to the Arduino outside:

Table 4. Sensors' pin definition.

Component	Digital/Analog Pin
Light Sensor	A0
Sound Sensor	6
Temperature Sensor	7
LED	2

3. Upload the Inside_Arduino sketch onto the Arduino outside.
4. Using the following table, connect the actuators to the Arduino inside:

Table 5. Actuators' pin definition.

Component	Digital/Analog Pin
LED	3
DC Motor	4
Servo Motor	9
Display	10

5. Connected both Arduinos to their respective edge devices.

5.2. Edge layer setup

6. Ensure that both edge devices connected to the same Wi-Fi network as the cloud server.
7. Change the MQTT broker IP address of these devices to that of the cloud server (line 16 on both outside_edge.py and inside_edge.py).
8. Create a database called sensorslog using mysql database service on the outside edge server.
9. Create a database called actuatorslog using mysql database service on the inside edge server.
10. Change the report schedule time in both the outside and inside edge to your desired time (line 243 on the outside_edge.py, and line 291 on the inside_edge.py).
11. Run the outside_edge.py script on the edge server outside; and inside_edge.py script on the edge server inside.

5.3. Cloud layer setup

12. Run the cloud_server.py script on the cloud server.
13. Open ThingsBoard and log in to access the system's dashboard. The username and password used for the demonstration is:
    - Username: nguyen@thingsboard.org
    - password: 12345678

6. Limitations

The known limitations are:

- Sound sensor is very sensitive so it will alert the user constantly.
- Light sensor does not record lux value but measure the current flowing through its resistor instead. This resistor will change value when receive light, which is used to estimate lux, but maybe inaccurate due to lux being a logarithmic unit instead of linear.
- Current implementation required the Arduinos to directly connected via wire to the edge server, which can be hard to do in reality, especially when the Arduino will be placed in remote areas like outside of the building.
- Current implementation required all the servers (inside edge, outside edge, and cloud) to connect to the same Wi-fi network. Thius can be hard to do if the cloud server is being hosted remotely on high-performance devices.
- Current dashboard doesn't have an alarm feature.
- Current dashboard contains minimal data visualization.
- No manual temperature and light threshold setting.

- Current dashboards receive data very slowly due to it having to travel through both layers (cloud and edge).
- No export function for the report generated on Discord.
- Discord webhook API doesn’t allow as many functionalities as creating a Discord bot.
- OpenWeather API is currently at free tier, restricting the amount of calls and types of data accessible.

7. Resources

[1] A. Yavari, Week 1, Slide 30, p. 15, Mar. 2025. [Online]. Available: https://swinburne.instructure.com/courses/66782/files/37376483?module_item_id=4942303

8. Appendix

8.1. Inside Arduino

```
#include <U8x8lib.h>
#include <Servo.h>

// ========== PIN DEFINITIONS ==========
#define LEDPIN 3
#define MOTORPIN 4
#define SERVOPIN 9

// ========== HARDWARE OBJECTS ==========
Servo doorServo;

// OLED Display Setup (Software I2C)
U8X8_SSD1306_128X64_NONAME_SW_I2C oledDisplay(SCL, SDA, U8X8_PIN_NONE);
int displayCols;
int displayRows;

// ========== DISPLAY MESSAGES ==========
const char* defaultMsg = "Ideal Conditions";
const char* lightMsg = "Turning on LED";
const char* fanMsg = "Turning on fan";
const char* doorMsg = "Open Door";

String screenMessage = defaultMsg; // Current message to display

// ========== CONTROL THRESHOLDS ==========
int lightLimit = 800;
float tempLimit = 30.0;

// ========== SYSTEM STATUS VARIABLES ==========
bool ack = false;
bool isManualMode = false;

// Actuator States
bool isLightOn = false;
bool isFanOn = false;
bool isDoorOpen = false;

// ========== SENSOR DATA VARIABLES ==========
String input = "";
String sensor = "";
float temp = 0.0;
int light = 0;
String sound = "";
bool isLoud = false;

// ========== TIMING CONTROL ==========
unsigned long previousMillis = 0;
const long updateInterval = 3000; // Update interval in milliseconds (3 seconds)
```

```
void setup() {
// Initialize serial communication
Serial.begin(9600);

// Configure digital pins
pinMode(LEDPIN, OUTPUT);
pinMode(MOTORPIN, OUTPUT); // Fan

// Initialize OLED display
oledDisplay.begin();
oledDisplay.setFont(u8x8_font_amstrad_cpc_extended_f);
oledDisplay.clear();



// Get display dimensions
displayCols = oledDisplay.getCols();
displayRows = oledDisplay.getRows();

// Show initial setup information
oledDisplay.setCursor(0, 0);
oledDisplay.print("Cols: " + String(displayCols));
oledDisplay.setCursor(0, 1);
oledDisplay.print("Rows: " + String(displayRows));

// Initialize servo to closed position
doorServo.attach(SERVOPIN);
doorServo.write(0); // Door closed

delay(2000); // Display setup info for 2 seconds
}



void loop() {
unsigned long currentMillis = millis();

// ========== PERIODIC STATUS OUTPUT ==========
// Send actuator status every 3 seconds
if (currentMillis - previousMillis >= updateInterval) {
previousMillis = currentMillis;

// Send actuator status via serial
Serial.print("ACTUATORS|"); // Actuator header
Serial.print("Mode: ");
Serial.print(isManualMode ? "manual" : "auto");
Serial.print(", Light: ");
Serial.print(isLightOn ? "on" : "off");
Serial.print(", Fan: ");
Serial.print(isFanOn ? "on" : "off");
Serial.print(", Door: ");
Serial.println(isDoorOpen ? "open" : "close");
// Update OLED display
displayMessage(screenMessage);
}
```

```
// ========== SERIAL COMMUNICATION HANDLING ==========
if (Serial.available() > 0) {
// Acknowledge receipt of data
ack = true;
Serial.print("SENSORS|"); // Sensor header
Serial.print("status: ");
Serial.println(ack ? "active" : "inactive");

// Read incoming data
input = Serial.readStringUntil('\n');
input.trim(); // Remove whitespace

// ========== COMMAND PARSING ==========
// Parse different types of incoming commands
if (input.startsWith("sensor:")) {
// Process sensor data from outdoor unit
parseSensorData(input);

} else if (input.startsWith("led:")) {
// Manual LED control
isLightOn = (input.substring(4) == "on");

} else if (input.startsWith("door:")) {
// Manual door control
isDoorOpen = (input.substring(5) == "open");



} else if (input.startsWith("fan:")) {
// Manual fan control
isFanOn = (input.substring(4) == "on");

} else if (input.startsWith("mode:")) {
// Switch between manual and automatic mode
isManualMode = (input.substring(5) == "manual");

} else if (input.startsWith("threshold:")) {
// Update temperature threshold
String valueStr = input.substring(10);
float valueFloat = valueStr.toFloat();
tempLimit = round(valueFloat);
}

// ========== AUTOMATIC CONTROL LOGIC ==========
if (!isManualMode) {
// Reset all actuator states
isLightOn = false;
isFanOn = false;
isDoorOpen = false;
screenMessage = "";

// Evaluate environmental conditions
bool isDay = light > lightLimit;
bool isHot = temp > tempLimit;
```

```
// LIGHTING CONTROL: Turn on LED when it's dark
isLightOn = !isDay;

// FAN CONTROL: Turn on fan when it's hot
isFanOn = isHot;

// DOOR CONTROL: Open door when conditions are favorable
// Door opens when: quiet AND (daylight + cool OR nighttime)
isDoorOpen = !isLoud && ((isDay && !isHot) || !isDay);

// ========== DISPLAY MESSAGE LOGIC ==========
// Determine what message to show based on active systems
if (!isLightOn && !isFanOn && !isDoorOpen) {
screenMessage = defaultMsg;
} else {
// Priority: Light > Fan > Door (only show one message)
if (isLightOn) screenMessage = lightMsg;
if (isFanOn) screenMessage = fanMsg;
if (isDoorOpen) screenMessage = doorMsg;
Serial.println();
}
}

// ========== ACTUATOR CONTROL ==========
// Apply the determined states to physical outputs
digitalWrite(LEDPIN, isLightOn ? HIGH : LOW);
digitalWrite(MOTORPIN, isFanOn ? HIGH : LOW);
doorServo.write(isDoorOpen ? 90 : 0); // 90° = open, 0° = closed
}

// Update display continuously (function handles change detection)
displayMessage(screenMessage);
}

// ========== DISPLAY MESSAGE FUNCTION ==========
// Updates the OLED display only when the message changes to avoid flicker.
// Centers the message both horizontally and vertically on the display.
void displayMessage(String message) {
static String previousMessage = ""; // Store last displayed message

// Only update display if message has changed (reduces flicker)
if (message != previousMessage) {
oledDisplay.clear();

// Calculate centered position
int messageCol = (displayCols - message.length()) / 2; // Horizontal center
int messageRow = displayRows / 2; // Vertical center

// Ensure cursor position is not negative
oledDisplay.setCursor(max(0, messageCol), messageRow);
oledDisplay.print(message);
previousMessage = message; // Remember current message
}
}
```

```
// ========== SENSOR DATA PARSER ==========
// Parses incoming sensor data string in format: "sensor:outside,temp:30,light:400,sound:Yes"
void parseSensorData(String data) {
int start = 0;

// Parse comma-separated key:value pairs
while (start < data.length()) {
int end = data.indexOf(',', start);
if (end == -1) end = data.length(); // Handle last pair

String pair = data.substring(start, end);
int sep = pair.indexOf(':');

if (sep != -1) {
String key = pair.substring(0, sep);
String value = pair.substring(sep + 1);

// Assign values based on key
if (key == "sensor") sensor = value;
else if (key == "temp") temp = value.toFloat();
else if (key == "light") light = value.toInt();
else if (key == "sound") isLoud = (value == "Yes");
}

start = end + 1; // Move to next pair
}
}
```

8.2. Outside Arduino

```
#include <DHT.h>

// ========== PIN DEFINITIONS ==========
#define LIGHTPIN A0
#define SOUNDPIN 6
#define DHTPIN 7
#define MSGINDICATORPIN 2

// ========== SENSOR CONFIGURATION ==========
#define DHTTYPE DHT22

// ========== SENSOR OBJECTS ==========
DHT dht(DHTPIN, DHTTYPE);

// ========== STATUS VARIABLES ==========
bool soundDetected = false;

// ========== COMMUNICATION VARIABLES ==========
String input = "";

// ========== TIMING CONTROL ==========
unsigned long previousMillis = 0;
const long updateInterval = 3000; // Sensor reading interval (3 seconds)
```

```
void setup() {
// Initialize serial communication at 9600 baud
Serial.begin(9600);

// Initialize DHT sensor
dht.begin();

// Configure digital pins
pinMode(SOUNDPIN, INPUT);
pinMode(MSGINDICATORPIN, OUTPUT);
}

void loop() {
unsigned long currentMillis = millis();

// ========== CONTINUOUS SOUND MONITORING ==========
// Check for sound detection continuously
// If sound is detected at any point during the 3-second interval,
// the soundDetected flag will be set to true
if (digitalRead(SOUNDPIN) == LOW) {
soundDetected = true;
}

// ========== PERIODIC SENSOR READING & TRANSMISSION ==========
// Every 3 seconds, read all sensors and transmit data
if (currentMillis - previousMillis >= updateInterval) {
previousMillis = currentMillis;

// ========== SENSOR READINGS ==========
// Read light sensor and convert to lux
int lux = calculateLux(analogRead(LIGHTPIN));

// Read temperature from DHT22 sensor
int temperature = round(dht.readTemperature());

// ========== DATA TRANSMISSION ==========
// Send sensor data in comma-separated format
Serial.print("Light:");
Serial.print(lux);
Serial.print(", Sound:");
Serial.print(soundDetected ? "Yes" : "No");
Serial.print(", Temperature:");
Serial.println(temperature);

// ========== RESET FLAGS ==========
// Reset sound detection flag for next interval
soundDetected = false;
}

// ========== ACKNOWLEDGMENT HANDLING ==========
// Check for incoming acknowledgment from indoor unit
if (Serial.available() > 0) {
input = Serial.readStringUntil('\n');
input.trim(); // Remove whitespace
```

```
// Parse acknowledgment status
if (input.startsWith("status:")) {
String ackValue = input.substring(7);
ackValue.trim(); // Remove whitespace

// Control indicator LED based on acknowledgment
if (ackValue == "active") {
digitalWrite(MSGINDICATORPIN, HIGH); // Turn on LED (message received)
} else if (ackValue == "inactive") {
digitalWrite(MSGINDICATORPIN, LOW); // Turn off LED
}
}
}
}

// ========== LUX CALCULATION FUNCTION ==========
int calculateLux(int analogValue) {
// Linear conversion: map 0-1023 to 0-6000 lux
float lux = analogValue * (6000.0 / 1023.0);
return round(lux); // Return rounded integer value
}
```

8.3. Cloud Edge Server

```
import json
import time
import threading
import requests
import paho.mqtt.client as mqtt
from datetime import datetime

#
================================================================================
# CONFIGURATION SECTION
#
================================================================================

# ThingsBoard Cloud Configuration
THINGSBOARD_BROKER = "mqtt.thingsboard.cloud"
THINGSBOARD_PORT = 1883
THINGSBOARD_TOKEN = "Edgeserver" # Device token for authentication

# ThingsBoard MQTT Topics
MQTT_SUBS_TB_TOPIC = "v1/devices/me/rpc/request/+"
MQTT_PUBS_TB_TOPIC = "v1/devices/me/telemetry"

# Local Edge Network Configuration
MQTT_SUBS_EDGE_TOPIC = ["edge/outside/data", "edge/inside/data"]
MQTT_PUBS_CLOUD_TOPIC_CONTROL = "cloud/control"
MQTT_PUBS_CLOUD_TOPIC_SUGGESTION = "cloud/suggestion"
```

```
LOCAL_BROKER = "172.20.10.14" # Change to cloud VM server address
LOCAL_PORT = 1883

# Weather API Configuration
OPENWEATHER_API_KEY = "your_api_key" # Replace with actual API key
LOCATION = "melbourne,au"

#
================================================================================
# GLOBAL STATE VARIABLES
#
================================================================================

# Inside environment state (actuators)
inside = {
    "fan": "off",
    "door": "close",
    "led": "off",
    "mode": "auto"
}

# Outside environment state (sensors)
outside = {
    "temperature": None,
    "light": None,
    "sound": None
}

# Weather information and decision parameters
weather = {
    "message": "",
    "temp": 0.0,
    "weather condition": "",
    "temp threshold": 30.0, # Default threshold
}

#
================================================================================
# MQTT CLIENT INITIALIZATION
#
================================================================================

# ThingsBoard MQTT client setup
tb_client = mqtt.Client()
tb_client.username_pw_set(THINGSBOARD_TOKEN)

# Local MQTT client setup
```

```
local_client = mqtt.Client()

#
================================================================================
# WEATHER DATA PROCESSING
#
================================================================================

def fetch_weather_loop():
    while True:
        try:
            # Fetch current weather data
            res = requests.get(
                f"http://api.openweathermap.org/data/2.5/weather?q={LOCATION}&
appid={OPENWEATHER_API_KEY}&units=metric"
            )
            data = res.json()

            # Extract temperature and weather condition
            temp = data["main"]["temp"]
            condition = data["weather"][0]["main"].lower()

            # Determine user message and temperature threshold based on
weather
            if condition in ["clear", "clouds"]:
                message = "◯ It's nice out! Go outside!"
                temp_threshold = 25.0
            elif condition in ["rain", "thunderstorm", "snow"]:
                message = "⛈ Weather's bad. Stay indoors!"
                temp_threshold = 35.0
            else:
                message = "😶‍🌫️ Mixed weather. Stay safe!"
                temp_threshold = 30.0

            # Update global weather state
            weather["message"] = message
            weather["temp"] = temp
            weather["weather condition"] = condition
            weather["temp threshold"] = temp_threshold

            print(f"[WEATHER] {message} | Outdoor Temp: {temp}°C, Condition:
{condition}")
            print(f"[DECISIONS] Temperature Threshold: {temp_threshold}")

        except Exception as e:
            print("[ERROR] Weather fetch failed:", e)

        # Wait 2 minutes before next fetch
```

```
        time.sleep(120)

#
================================================================================
# DATA PUBLISHING FUNCTIONS
#
================================================================================

def publish_to_thingsboard():
    while True:
        # Create combined telemetry payload
        payload = {
            "timestamp": datetime.now().isoformat(),
            **inside,
            **outside,
            **weather
        }

        try:
            tb_client.publish(MQTT_PUBS_TB_TOPIC, json.dumps(payload))
            print("[TB] Published:", payload)
        except Exception as e:
            print("[TB ERROR] Publish failed:", e)

        # Publish every 10 seconds
        time.sleep(10)

def publish_weather():
    while True:
        try:
            weather_payload = json.dumps(weather)
            local_client.publish(MQTT_PUBS_CLOUD_TOPIC_SUGGESTION,
weather_payload)
            print(f"[FORWARD] Pubslished to
{MQTT_PUBS_CLOUD_TOPIC_SUGGESTION}:", weather_payload)

        except Exception as e:
            print("[ERROR] publish_weather failed:", e)

        # Publish every 2 minutes
        time.sleep(120)

#
================================================================================
# THINGSBOARD MQTT HANDLERS
#
================================================================================
```

```
def tb_on_connect(client, userdata, flags, rc):
    print("[TB] Connected")
    client.subscribe(MQTT_SUBS_TB_TOPIC)

def tb_on_message(client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode())
        print("[TB] RPC received:", payload)

        # Extract RPC method and parameters
        method = payload.get("method")
        params = payload.get("params")

        if method:
            # Update local state
            inside[method] = params
            print(f"[RPC] {method} set to {params}")

            # Forward command to edge layer via local MQTT
            command_payload = json.dumps({method: params}) # e.g. {"led" :
"on"}
            command_topic = f"{MQTT_PUBS_CLOUD_TOPIC_CONTROL}/{method}" # e.g.
"cloud/control/led"
            local_client.publish(command_topic, command_payload)
            print(f"[FORWARD] Pubslished to {command_topic}:",
command_payload)

    except Exception as e:
        print("[TB ERROR] on_message:", e)

#
================================================================================
# LOCAL MQTT HANDLERS
#
================================================================================

def local_on_connect(client, userdata, flags, rc):
    print("[LOCAL] Connected")
    for topic in MQTT_SUBS_EDGE_TOPIC:
        client.subscribe(topic)

def local_on_message(client, userdata, msg):
    topic = msg.topic
    try:
        data = json.loads(msg.payload.decode())
        print(f"[LOCAL] {topic} -> {data}")

        # Update inside actuator states
```

```
        if "inside" in topic:
            for key in ["fan", "door", "led", "mode"]:
                if key in data:
                    inside[key] = data[key]

        # Update outside sensor readings
        elif "outside" in topic:
            for key in ["temperature", "light", "sound"]:
                if key in data:
                    outside[key] = data[key]
    except Exception as e:
        print("[LOCAL ERROR]", e)

#
================================================================================
# MAIN EXECUTION
#
================================================================================

# Configure MQTT client callbacks
tb_client.on_connect = tb_on_connect
tb_client.on_message = tb_on_message

local_client.on_connect = local_on_connect
local_client.on_message = local_on_message

# Establish MQTT connections
tb_client.connect(THINGSBOARD_BROKER, THINGSBOARD_PORT, 60)
tb_client.loop_start()

local_client.connect(LOCAL_BROKER, LOCAL_PORT, 60)
local_client.loop_start()

# Start background threads
threading.Thread(target=fetch_weather_loop, daemon=True).start()
threading.Thread(target=publish_to_thingsboard, daemon=True).start()
threading.Thread(target=publish_weather, daemon=True).start()

# Keep main thread alive
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    tb_client.loop_stop()
    local_client.loop_stop()
    print("Stopped.")
```

8.4. Inside Edge Server

```
import serial
import pymysql
from datetime import datetime
import time
import threading
import paho.mqtt.client as mqtt
import json
import requests
import schedule

#
================================================================================
# CONFIGURATION SECTION
#
================================================================================

# MQTT Configuration
MQTT_BROKER = "172.20.10.14" # Change to cloud VM server address
MQTT_SUBS_EDGE_TOPIC = "edge/outside/data"
MQTT_PUBS_EDGE_TOPIC = "edge/outside/status"
MQTT_SUBS_CLOUD_TOPIC_CONTROL = "cloud/control/#"
MQTT_SUBS_CLOUD_TOPIC_SUGGESTION = "cloud/suggestion"
MQTT_PUBS_CLOUD_TOPIC = "edge/inside/data"

# Discord Integration
DISCORD_WEBHOOK_URL =
"https://discord.com/api/webhooks/1375385948715487243/18tL62HUw6PFjRXGYorL1Age
2WsKibXKvwc5zlJGQCLdNlp9O6B6cBvC9tg_grTRz9_O"

#
================================================================================
# GLOBAL VARIABLES
#
================================================================================

# Track last actuator states for change detection
last_state = {"led": None, "door": None, "fan": None}

#
================================================================================
# HARDWARE INITIALIZATION
#
================================================================================

# Initialize Arduino serial connection
arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
```

```
# Initialize MQTT client
MQTT_CLIENT = mqtt.Client()
MQTT_CLIENT.connect(MQTT_BROKER, 1883, 60)

#
================================================================================
# MQTT EVENT HANDLERS
#
================================================================================

def on_connect(client, userdata, flags, rc):
    print(f"[MQTT] Connected with result code {rc}")
    client.subscribe(MQTT_SUBS_EDGE_TOPIC)
    client.subscribe(MQTT_SUBS_CLOUD_TOPIC_CONTROL)
    client.subscribe(MQTT_SUBS_CLOUD_TOPIC_SUGGESTION)

def on_message(client, userdata, msg):
    global current_mode
    topic = msg.topic
    payload_str = msg.payload.decode()
    print(f"[MQTT] Message received: {topic} -> {payload_str}")

    try:
        # Handle outside sensor data
        if topic == MQTT_SUBS_EDGE_TOPIC:
            payload = json.loads(payload_str)
            temp = payload["temperature"]
            light = payload["light"]
            sound = payload["sound"]

            # Forward sensor data to Arduino
            send_to_arduino(f"sensor:outside,temp:{temp},light:{light},sound:{
sound}")

        # Handle mode control commands
        elif topic == "cloud/control/mode":
            payload = json.loads(payload_str)
            new_mode = payload.get("mode","").lower()

            if new_mode in ["auto", "manual"]:
                current_mode = new_mode
                send_discord_alert(f"⚙️⚙️ CONTROL MODE changed to
{current_mode.upper()} ⚙️⚙️")

            # Send mode update to Arduino
            send_to_arduino(f"mode:{current_mode}")

        # Handle actuator control commands
```

```
        elif topic.startswith("cloud/control/"):
            # Check if system is in auto mode
            if current_mode == "auto":
                print("[INFO] Ignoring actuator command in AUTO mode.")
                send_discord_alert("⚠⚠ WARING: SYSTEM in AUTO MODE, IGNORED
COMMAND ⚠⚠")
                return  # Ignore command in auto mode

            # Extract actuator type from topic
            actuator = topic.split("/")[-1]
            payload = json.loads(payload_str)
            value = str(payload.get(f"{actuator}", "")).lower()

            # Send command to Arduino
            command = f"{actuator}:{value}"
            send_to_arduino(command)
            handle_actuator_command(command)

        # Handle cloud weather suggestions
        elif topic == MQTT_SUBS_CLOUD_TOPIC_SUGGESTION:
            payload = json.loads(payload_str)
            message = payload.get("message", "")

            # Send weather message to Discord
            send_discord_alert(f"🌤🌤 MESSAGE FROM CLOUD: {message}🌤🌤")

            # Update temperature threshold
            temp_threshold = payload.get("temp threshold", 30.0)
            temp_threshold_str = f"threshold:{temp_threshold}"
            send_to_arduino(temp_threshold_str)

    except Exception as e:
        print("[ERROR] on_message:", e)

#
================================================================================
# SERIAL COMMUNICATION FUNCTIONS
#
================================================================================

def send_to_arduino(message: str):
    try:
        arduino.write((message + '\n').encode())
        print(f"[Serial] Sent to Arduino: {message}")
    except Exception as e:
        print("[ERROR] Sending to Arduino:", e)

def handle_actuator_command(cmd: str):
```

```
    global last_state
    try:
        key, value = cmd.split(':')
        if key in last_state and last_state[key] != value:
            send_discord_alert(f"🔄🔄 ACTUATOR '{key.upper()}' CHANGED TO:
{value.upper()} 🔄🔄")
            last_state[key] = value
    except Exception as e:
        print("[ERROR] handle_actuator_command:", e)

#
================================================================================
# NOTIFICATION FUNCTIONS
#
================================================================================

def send_discord_alert(message):
    data = {"content": message}
    try:
        requests.post(DISCORD_WEBHOOK_URL, json=data)
    except Exception as e:
        print("[ERROR] Discord alert failed:", e)

#
================================================================================
# DATABASE FUNCTIONS
#
================================================================================

def get_db_connection(host='localhost', user='root', password='12345678',
db='actuatorslog'):
    try:
        conn = pymysql.connect(host=host, user=user, password=password,
database=db)
        cur = conn.cursor()

        # Create table if it doesn't exist
        cur.execute('''
            CREATE TABLE IF NOT EXISTS logs (
                time DATETIME,
                led VARCHAR(20),
                fan VARCHAR(20),
                door VARCHAR(20),
                mode VARCHAR(20)
            )
        ''')
        conn.commit()
        return conn
```

```
    except Exception as e:
        print(f"[ERROR] Database connection failed: {e}")
        return None

def log_data():
    while True:
        try:
            if arduino.in_waiting:
                msg = arduino.readline().decode().strip()

                # Process actuator status messages
                if msg.startswith("ACTUATORS|"):
                    parts = msg.split(',')
                    current_mode = parts[0].split(':')[1]
                    led = parts[1].split(':')[1]
                    fan = parts[2].split(':')[1]
                    door = parts[3].split(':')[1]
                    now = datetime.now()

                    # Log to database
                    conn = get_db_connection()
                    cur = conn.cursor()
                    cur.execute("INSERT INTO logs (time, led, fan, door, mode)
VALUES (%s, %s, %s, %s, %s)",
                                (now, led, fan, door, current_mode))
                    conn.commit()
                    conn.close()

                    # Publish to MQTT
                    payload = json.dumps({
                        "time": now.isoformat(),
                        "led": led,
                        "fan": fan,
                        "door": door,
                        "mode": current_mode
                    })
                    MQTT_CLIENT.publish(MQTT_PUBS_CLOUD_TOPIC, payload)
                    print(f"[MQTT] Published: {payload} to
{MQTT_PUBS_CLOUD_TOPIC}")

                # Process sensor acknowledgment messages
                elif msg.startswith("SENSORS|"):
                    parts = msg.split(',')
                    ack = parts[0].split(':')[1]
                    payload = json.dumps({ "sensors": ack })
                    MQTT_CLIENT.publish(MQTT_PUBS_EDGE_TOPIC, payload)
                    print(f"[MQTT] Published: {payload} to
{MQTT_PUBS_EDGE_TOPIC}")
```

```
        except Exception as e:
            print("[ERROR] log_data:", e)
        time.sleep(1)

#
================================================================================
# REPORTING FUNCTIONS
#
================================================================================

def generate_reports():
    try:
        print("[INFO] Generating Report")
        conn = get_db_connection()
        cursor = conn.cursor()

        # Define today's time range
        today = datetime.now().date()
        start_time = datetime.combine(today, datetime.min.time())
        end_time = datetime.combine(today, datetime.max.time())

        # Query actuator data for today
        cursor.execute("""
            SELECT led, fan, door, mode
            FROM logs
            WHERE time BETWEEN %s AND %s
            ORDER BY time ASC
        """, (start_time, end_time))
        actuator_rows = cursor.fetchall()

        # Generate report content
        actuator_report = "**ACTUATORS REPORT**\n"
        if not actuator_rows:
            actuator_report += "No actuator activity recorded today.\n"
        else:
            # Count state transitions
            led_count = door_count = fan_count = mode_count = 0
            last_led = last_door = last_fan = last_mode = None

            for led, servo, fan, mode in actuator_rows:
                # Count on transitions
                if last_led is not None and last_led == " off" and led == "
on":
                    led_count += 1
                if last_door is not None and last_door == " closed" and servo
== " open":
                    door_count += 1
```

```
                if last_fan is not None and last_fan == " off" and fan == "
on":
                    fan_count += 1
                if last_mode is not None and last_mode == " manual" and mode
== " auto":
                    mode_count += 1
                last_led, last_door, last_fan, last_mode = led, servo, fan,
mode

            actuator_report += f"LED turned ON: {led_count} times\n"
            actuator_report += f"Door opened: {door_count} times\n"
            actuator_report += f"Fan turned ON: {fan_count} times\n"
            actuator_report += f"Mode changed to MANUAL: {mode_count} times\n"

        send_discord_report("⚙ Daily Actuator Report", actuator_report)

        cursor.close()
        conn.close()

    except Exception as e:
            print("[ERROR] generate_report: ", e)

def send_discord_report(title, content):
    data = {
        "embeds": [
            {
                "title": title,
                "description": content,
                "color": 5814783
            }
        ]
    }
    requests.post(DISCORD_WEBHOOK_URL, json=data)

def schedule_report():
    # Schedule task to run at specific time daily
    schedule_time = "23:59"
    schedule.every().day.at(schedule_time).do(generate_reports)
    print(f"Scheduler started. Waiting for {schedule_time} every day...")
    while True:
        schedule.run_pending()
        time.sleep(10)

#
================================================================================
# MAIN EXECUTION
#
================================================================================
```

```
# Configure MQTT client
MQTT_CLIENT.on_message = on_message
MQTT_CLIENT.on_connect = on_connect
MQTT_CLIENT.loop_start()

# Start background threads
threading.Thread(target=log_data, daemon=True).start()
threading.Thread(target=schedule_report, daemon=True).start()

# Keep main thread alive
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    MQTT_CLIENT.loop_stop()
    MQTT_CLIENT.disconnect()
```

8.5. Outside Edge Server

```
import serial
import pymysql
from datetime import datetime
import time
import threading
import paho.mqtt.client as mqtt
import json
import requests
import schedule

#
================================================================================
# CONFIGURATION SECTION
#
================================================================================

# MQTT Configuration
MQTT_BROKER = "172.20.10.14" # Change to cloud VM server address
MQTT_PUBS_TOPIC = "edge/outside/data"
MQTT_SUBS_TOPIC = ["edge/outside/status", "cloud/suggestion"]

# Discord Integration
DISCORD_WEBHOOK_URL =
"https://discord.com/api/webhooks/1375385948715487243/18tL62HUw6PFjRXGYorL1Age
2WsKibXKvwc5zlJGQCLdNlp9O6B6cBvC9tg_grTRz9_O"

# Sensor Thresholds
LIGHT_THRESHOLD = 800
TEMP_THRESHOLD = 30.0
```

```
#
# ============================================================================
# GLOBAL VARIABLES
#
# ============================================================================

# Store previous sensor states for edge detection
prev_sound = "no"
prev_light_exceeded = False
prev_temp_exceeded = False

#
# ============================================================================
# HARDWARE INITIALIZATION
#
# ============================================================================

# Initialize Arduino serial connection
arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=1)

# Initialize MQTT client
MQTT_CLIENT = mqtt.Client()
MQTT_CLIENT.connect(MQTT_BROKER, 1883, 60)

#
# ============================================================================
# DATABASE FUNCTIONS
#
# ============================================================================

def get_db_connection(host='localhost', user='root', password='12345678',
db='sensorslog'):
    try:
        conn = pymysql.connect(host=host, user=user, password=password,
database=db)
        cur = conn.cursor()

        # Create table if it doesn't exist
        cur.execute('''
            CREATE TABLE IF NOT EXISTS logs (
                time DATETIME,
                light INT,
                sound VARCHAR(20),
                temperature INT
            )
        ''')
        conn.commit()
```

```
        return conn
    except Exception as e:
        print(f"[ERROR] Database connection failed: {e}")
        return None

#
================================================================================
# DATA PROCESSING FUNCTIONS
#
================================================================================

def log_and_publish_data():
    global prev_sound, prev_light_exceeded, prev_temp_exceeded
    while True:
        try:
            if arduino.in_waiting:
                # Read sensor data from Arduino
                line = arduino.readline().decode().strip()
                parts = line.split(',')

                # Parse sensor values
                light = int(parts[0].split(':')[1])
                sound = parts[1].split(':')[1]
                temp = int(parts[2].split(':')[1])
                now = datetime.now()

                # SOUND ALERT: Rising edge detection (quiet -> loud)
                if sound == "yes" and prev_sound != "yes":
                    send_discord_alert("🔊🔊 SOUND changed to LOUD, CAUTION
🔊🔊")
                prev_sound = sound

                # LIGHT ALERT: Rising edge detection (normal -> bright)
                light_exceeded = light > LIGHT_THRESHOLD
                if light_exceeded and not prev_light_exceeded:
                    send_discord_alert(f"💡💡 BRIGHTNESS changed to {light}
lux, EXCEEDED {LIGHT_THRESHOLD} lux, CAUTION 💡💡")
                prev_light_exceeded = light_exceeded

                # TEMPERATURE ALERT: Rising edge detection (normal -> hot)
                temp_exceeded = temp > TEMP_THRESHOLD
                if temp_exceeded and not prev_temp_exceeded:
                    send_discord_alert(f"🔥🔥 TEMPERATURE change to {temp}
°C, EXCEEDED {TEMP_THRESHOLD} °C, CAUTION 🔥🔥")
                prev_temp_exceeded = temp_exceeded

                # Save sensor data to database
                conn = get_db_connection()
```

```
                cur = conn.cursor()
                cur.execute(
                    "INSERT INTO logs (time, light, sound, temperature) VALUES
(%s, %s, %s, %s)",
                    (now, light, sound, temp)
                )
                conn.commit()
                conn.close()

                # Prepare payload and publish to MQTT
                payload = json.dumps({
                    "timestamp": now.isoformat(),
                    "light": light,
                    "sound": sound,
                    "temperature": temp
                })
                MQTT_CLIENT.publish(MQTT_PUBS_TOPIC, payload)
                print(f"[INFO] Published: {payload} to {MQTT_PUBS_TOPIC}")

        except Exception as e:
            print("[Error] Sending to Arduino or MQTT pusblishing:", e)
        time.sleep(1)

#
================================================================================
# NOTIFICATION FUNCTIONS
#
================================================================================

def send_discord_alert(message):
    data = {"content": message}
    try:
        requests.post(DISCORD_WEBHOOK_URL, json=data)
    except Exception as e:
        print("[Error] Failed to send Discord alert:", e)

#
================================================================================
# MQTT EVENT HANDLERS
#
================================================================================

def on_connect(client, userdata, flags, rc):
    print(f"[MQTT] Connected with result code {rc}")
    for topic in MQTT_SUBS_TOPIC:
        client.subscribe(topic)

def on_message(client, userdata, msg):
```

```
    topic = msg.topic
    payload_str = msg.payload.decode()
    print(f"[MQTT] Message received: {topic} -> {payload_str}")

    try:
        # Handle edge server data
        if "edge" in topic:
            payload = json.loads(payload_str)
            ack = payload["sensors"]
            send_to_arduino(f"status:{ack}")

        # Handle cloud server data
        elif "cloud" in topic:
            global TEMP_THRESHOLD
            payload = json.loads(payload_str)
            TEMP_THRESHOLD = payload.get("temp threshold", 30.0)

    except Exception as e:
        print("[Error] on_message:", e)

#
# ============================================================================
# SERIAL COMMUNICATION FUNCTIONS
#
# ============================================================================

def send_to_arduino(message: str):
    try:
        arduino.write((message + '\n').encode())
        print(f"[Serial] Sent to Arduino: {message}")
    except Exception as e:
        print("[Error] Sending to Arduino:", e)

#
# ============================================================================
# REPORTING FUNCTIONS
#
# ============================================================================

def generate_reports():
    try:
        print("[INFO] Generating Report")
        conn = get_db_connection()
        cursor = conn.cursor()

        # Define today's time range
        today = datetime.now().date()
        start_time = datetime.combine(today, datetime.min.time())
```

```
        end_time = datetime.combine(today, datetime.max.time())

        # Query sensor data for today
        cursor.execute("""
            SELECT light, sound, temperature
            FROM logs
            WHERE time BETWEEN %s AND %s
        """, (start_time, end_time))
        sensor_rows = cursor.fetchall()

        # Generate report content
        sensor_report = "**SENSORS DAILY REPORT**\n"
        total = len(sensor_rows)
        if total == 0:
            sensor_report += "No sensor data recorded today.\n"
        else:
            # Only add if conditions are met
            light_high = sum(1 for l, _, _ in sensor_rows if l >
LIGHT_THRESHOLD)
            sound_high = sum(1 for _, s, _ in sensor_rows if s == "yes")
            temp_high = sum(1 for _, _, t in sensor_rows if t >
TEMP_THRESHOLD)

            sensor_report += f"Total records: {total}\n"
            sensor_report += f"Light > {LIGHT_THRESHOLD} lux: {light_high /
total * 100:.2f}% of time\n"
            sensor_report += f"Loud noise: {sound_high / total * 100:.2f}% of
time\n"
            sensor_report += f"Temperature > {TEMP_THRESHOLD}°C: {temp_high /
total * 100:.2f}% of time\n"

        send_discord_report("📊 Daily Sensor Report", sensor_report)

        cursor.close()
        conn.close()
    except Exception as e:
        print("[Error] generate report", e)

def send_discord_report(title, content):
    data = {
        "embeds": [
            {
                "title": title,
                "description": content,
                "color": 5814783
            }
        ]
    }
```

```
    requests.post(DISCORD_WEBHOOK_URL, json=data)

def schedule_report():
    # Schedule task to run at specific time daily
    schedule_time = "23:59"
    schedule.every().day.at(schedule_time).do(generate_reports)
    print(f"Scheduler started. Waiting for {schedule_time} every day...")
    while True:
        schedule.run_pending()
        time.sleep(10)

#
================================================================================
# MAIN EXECUTION
#
================================================================================

# Configure MQTT client
MQTT_CLIENT.on_message = on_message
MQTT_CLIENT.on_connect = on_connect
MQTT_CLIENT.loop_start()

# Start background threads
threading.Thread(target=log_and_publish_data, daemon=True).start()
threading.Thread(target=schedule_report, daemon=True).start()

# Keep main thread alive
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    MQTT_CLIENT.loop_stop()
    MQTT_CLIENT.disconnect()
```