

**DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION ENGINEERING
UNIVERSITY OF MORATUWA**



EN3251 – Internet of Things

Lab Assignment 1: Implementing an end-to-end IoT system with JSON and MQTT

TEAM ENIGMA

**HIMEKA S. H. D. 200222K
NAZAR F. S. 200417M
RATHNAYAKE R.N.P. 200537F**

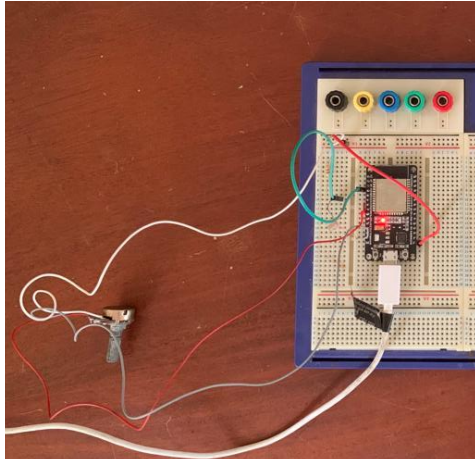
OCTOBER 1, 2023

Assignment Part A

Task 1:

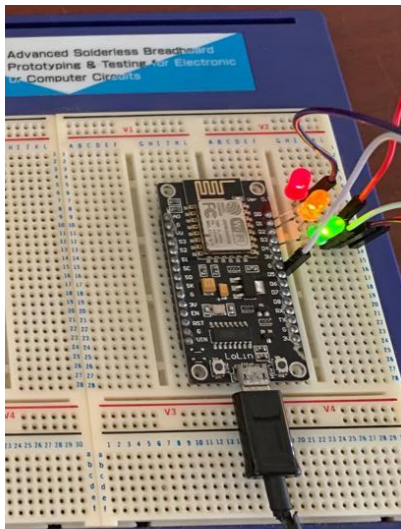
Creating an IoT node (Node A) consisting of an LDR and a Potentiometer as sensors as inputs.

Here, we used an ESP32 microcontroller and interfaced it with an LDR and a potentiometer through its analog input pins.



Creating an IoT node (Node B) consisting of three LEDs as outputs (Physical LED Dashboard).

Here, we used an ESP8266 microcontroller and interfaced it with three LEDs, red, green and orange through its digital output pins.



Task 2:

Designing the IoT system

At Node A:

The microcontroller reads the values of the LDR and potentiometer, which can range from 0 to 4095. It then assigns them to one of the categories: high, medium, or low, based on their values. Subsequently, these categorized values are encapsulated within a JSON object and published to an MQTT broker ("broker.hivemq.com") with the topic "EnigmaTaskA." This IoT node consistently publishes a JSON object at 5-second intervals.

```
StaticJsonDocument<256> doc;
int potValue=analogRead(potPin);
int ldrValue=analogRead(ldrPin);

doc["PotValue"] = potValue;
doc["LDRValue"] = ldrValue;
doc["PotRange"] = range(potValue);
doc["LDRRange"] = range(ldrValue);
serializeJson(doc, out);

// convert integer port value to string, all data should be string in mqtt
char valueStr[5];
snprintf(valueStr, sizeof(valueStr), "%d", potValue);

if(client.connected()){
  client.publish(mqttTopic, out); //publish the value(string) valueStr to the topic mqttTopic
  Serial.println("Published to MQTT: "+ String(out));
  delay(5000);
}
```

We decided the ranges of the LDR and Potentiometer values according to the below function.

```
String range (int Value){
  if (Value>=2730){
    return "High";
  }
  else if (Value < 2730 && Value >= 1365 ){
    return "Medium";
  }
  else {
    return "Low" ;
  }
}
```

The Json object which is send by node A has the following format.

```
{ "PotValue":190, "LDRValue":300, "PotRange":"Low", "LDRRange":"Low" }
```

At Node B:

This node functions as an MQTT subscriber, subscribing to the topic 'EnigmaTaskA.' It receives a JSON object containing the range values of the LDR and Pot from the MQTT broker. Subsequently, based on these received values, the microcontroller controls the blinking of the LEDs.

The following code defines a helper function blink() to blink all LEDs at the same time.

```
void blink(int r, int o, int g) {  
    digitalWrite(r, HIGH);  
    digitalWrite(o, HIGH);  
    digitalWrite(g, HIGH);  
  
    delay(1000);  
  
    digitalWrite(r, LOW);  
    digitalWrite(o, LOW);  
    digitalWrite(g, LOW);  
    delay(1000);  
}
```

The following callback() function is invoked whenever there is an incoming MQTT message on the subscribed topic. It parses the received JSON payload and extracts the values for "potValue" and "ldrValue," storing them in character arrays potValue and ldrValue.

```
void callback(char* topic, byte* payload, unsigned int length) {  
    char payloadStr[length + 1];  
    memcpy(payloadStr, payload, length);  
    payloadStr[length] = '\0';  
  
    StaticJsonDocument<32> doc;  
    DeserializationError error = deserializeJson(doc, payloadStr);  
  
    // Use strcpy to copy the values from the JSON object to your character arrays  
  
    if (error) {  
        Serial.print("Error parsing JSON: ");  
        Serial.println(error.c_str());  
        return;  
    }  
  
    if (doc.containsKey("potValue") && doc.containsKey("ldrValue")) {  
        strcpy(potValue, doc["potValue"], sizeof(potValue));  
        strcpy(ldrValue, doc["ldrValue"], sizeof(ldrValue));  
        Serial.print("Pot Value: ");  
        Serial.println(potValue);  
        Serial.print("LDR Value: ");  
        Serial.println(ldrValue);  
    }  
  
    else {  
        Serial.println("JSON object is missing keys.");  
    }  
}
```

In the loop() function, it continuously calls client.loop() to maintain MQTT communication. It then checks the values of potValue and ldrValue and controls the LEDs based on certain conditions:

- If both values are "High" or one is "High" and the other is "Medium," all LEDs are turned on.
- If both values are "Medium" or one is "High" and the other is "Low," the orange and green LEDs are turned on.
- If both values are "Low" or one is "Low" and the other is "Medium," the LEDs are made to blink using the blink() function.

```
void loop() {
  client.loop();

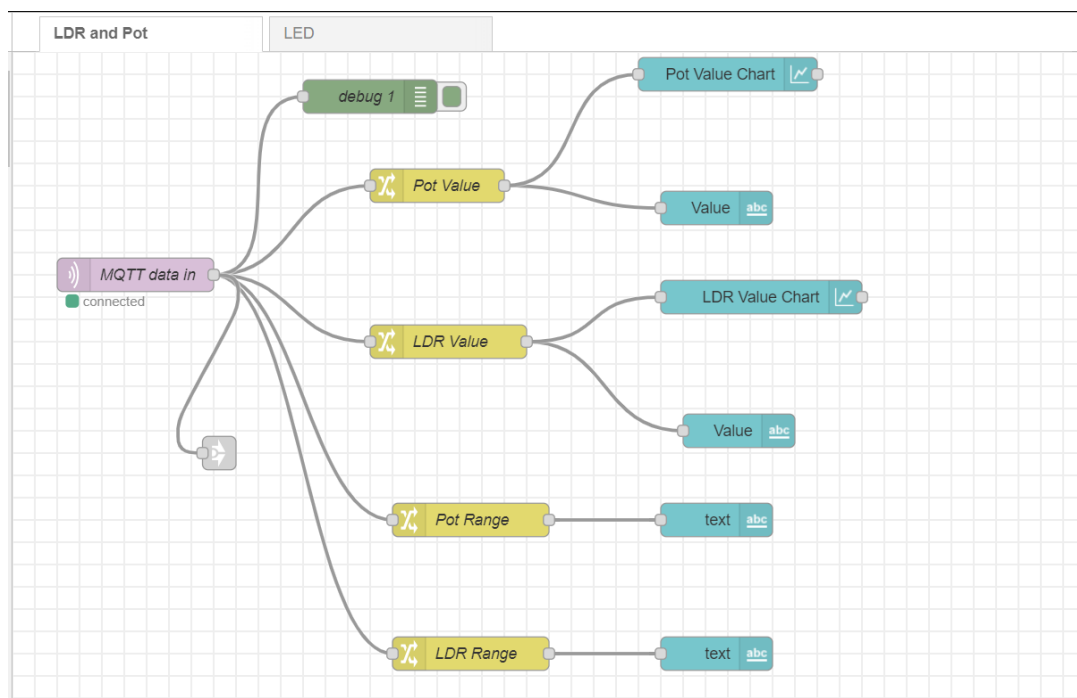
  if ((strcmp(potValue, "High") == 0 && strcmp(ldrValue, "Medium") == 0) || (strcmp(potValue, "Medium") == 0 && strcmp(ldrValue, "High") == 0) || (strcmp(potValue, "High") == 0 && strcmp(ldrValue, "High") == 0)) {
    // Turn on the red LED
    digitalWrite(redLED, HIGH);
    digitalWrite(orangeLED, HIGH);
    digitalWrite(greenLED, HIGH);
  }
  else if ((strcmp(potValue, "Medium") == 0 && strcmp(ldrValue, "Medium") == 0) || (strcmp(potValue, "High") == 0 && strcmp(ldrValue, "Low") == 0) || (strcmp(potValue, "Low") == 0 && strcmp(ldrValue, "High") == 0)) {
    // Turn on the orange LED
    digitalWrite(redLED, LOW);
    digitalWrite(orangeLED, HIGH);
    digitalWrite(greenLED, HIGH);
  }
  else if ((strcmp(potValue, "Low") == 0 && strcmp(ldrValue, "Low") == 0) || (strcmp(potValue, "Low") == 0 && strcmp(ldrValue, "Medium") == 0) || (strcmp(potValue, "Medium") == 0 && strcmp(ldrValue, "Low") == 0)) {
    blink(redLED, orangeLED, greenLED);
  }
}
```

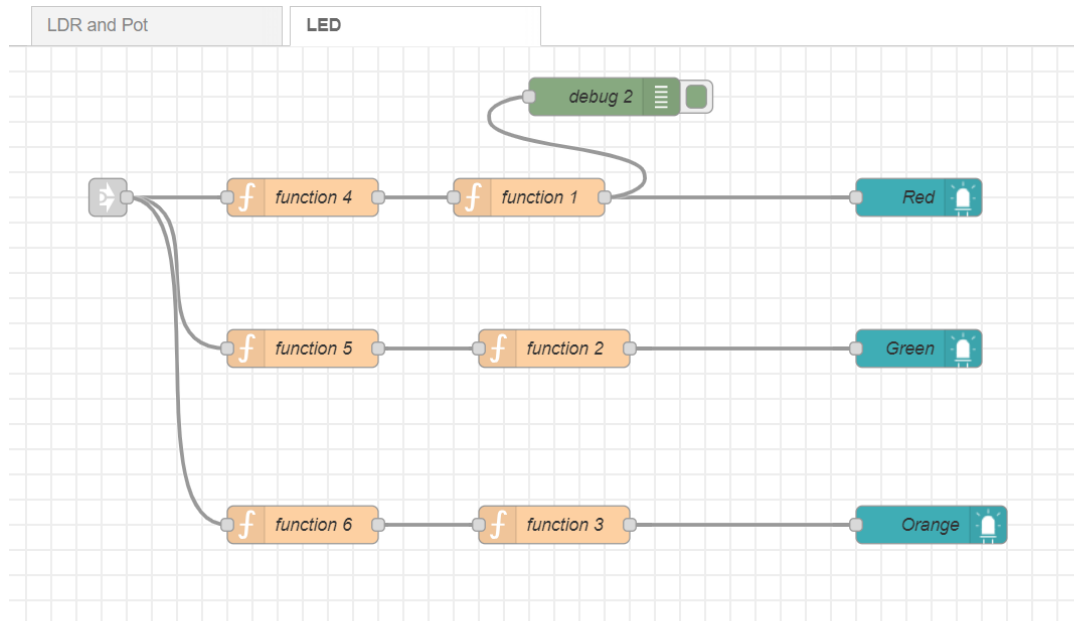
These codes essentially listen for sensor data over MQTT and respond by controlling LEDs based on the received data.

Task 3:

NodeRed Dashboard

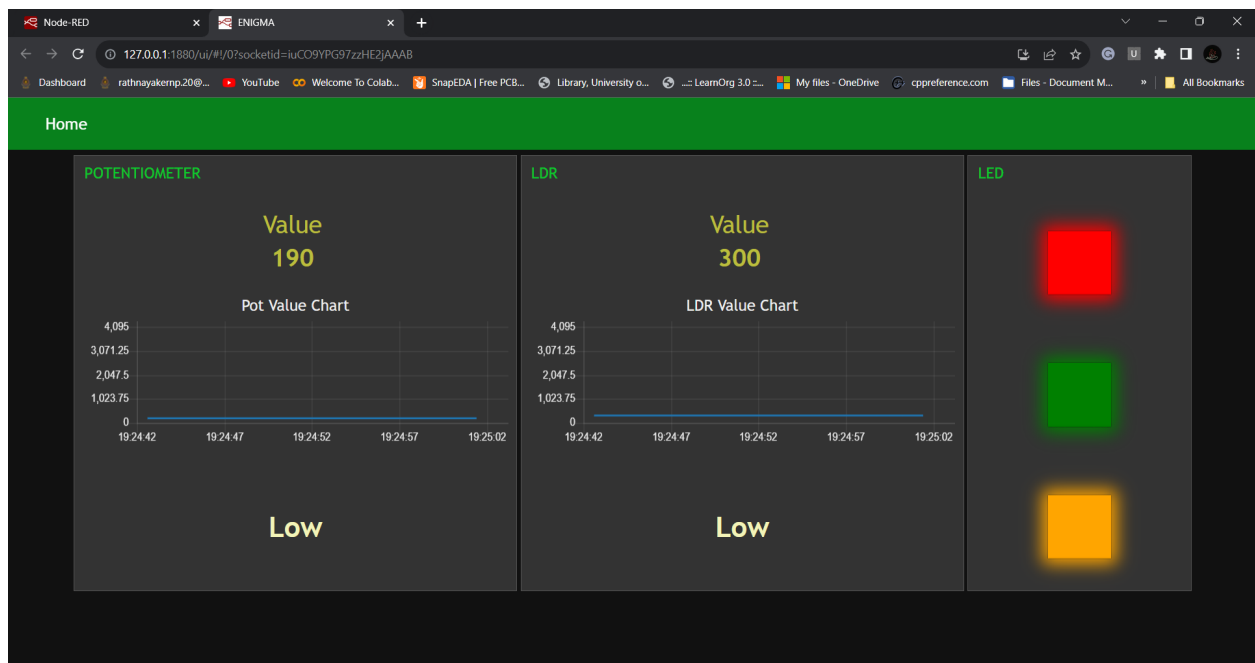
- The Node Red Flow





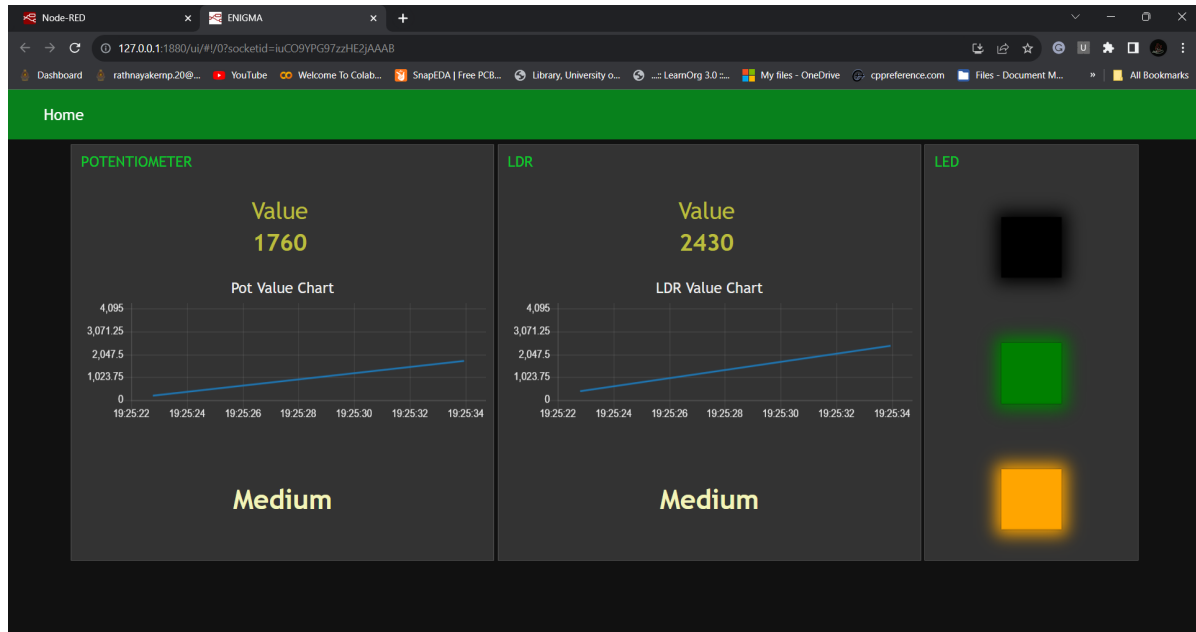
- The published message:
`{ "PotValue":190, "LDRValue":300, "PotRange":"Low", "LDRRange":"Low" }`

In this case, the LEDs blink.



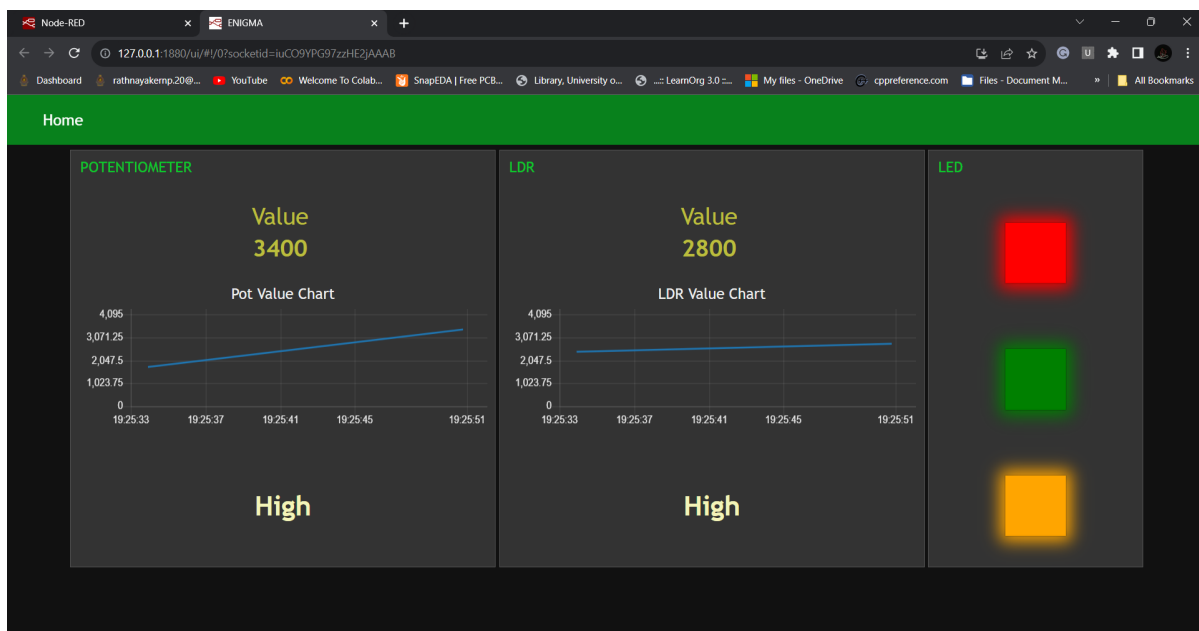
- The published message:
`{ "PotValue":1760, "LDRValue":2430, "PotRange":"Medium", "LDRRange":"Medium" }`

In this case, only the red led is off and others are on.



- The published message:
`{ "PotValue":3400, "LDRValue":2800, "PotRange":"High", "LDRRange":"High" }`

In this case, all LEDs are on.



The two charts show the values of the LDR and potentiometer for the past 20 seconds.

Assignment Part B:

Given CSV file:

	A	B	C	D	E
1	Location	Temperature	Humidity	Light	
2	Living room	20	10	250	
3	Kitchen	25	30	200	
4	Bed room	18	60	150	
5	Garage	30	40	70	
6					
7					
8					
9					

Task 1:

Publish the entire contents of the worksheet as a single JSON object to a single topic in an MQTT broker

```
import paho.mqtt.client as mqtt
import time
import json
import pandas as pd

df = pd.read_excel('data.xlsx')
json_data = df.to_dict(orient='records')
json_object = {str(i): record for i, record in enumerate(json_data)}
data_out = json.dumps(json_object)

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
    else:
        print("Connection failed with code {rc}")

# Create an MQTT client instance
client = mqtt.Client("PythonPub")

# Set the callback function
client.on_connect = on_connect

broker_address = "test.mosquitto.org" # broker's address
broker_port = 1883
```



```

keepalive = 5
qos = 0
publish_topic = "Enigma-HomeConditions"

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

# Start the MQTT loop to handle network traffic
client.loop_start()

# Publish loop
try:
    while True:
        # Publish a message to the send topic
        client.publish(publish_topic, data_out)
        print(f"Published message '{data_out}' to topic '{publish_topic}'\n")

        # Wait for a moment to simulate some client activity
        time.sleep(6)

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")

```

This Python code extracts data from an CSV file, converts it to JSON format, and delivers it to a MQTT broker using the MQTT protocol on a regular basis. The code connects to the broker, publishes the data to a specific topic("Enigma-HomeConditions"), and then waits a few seconds before repeating the process. Speciality is sending all csv data at once as a JSON object. It handles graceful program termination and MQTT broker disconnection. Essentially, it is a simple program that uses MQTT to share data with other devices or systems.

Results:

```


Connected to MQTT broker


Published message '{"0": {"Location": "Living room", "Temperature": 20, "Humidity": 10, "Light": 250}, "1": {"Location": "Kitchen", "Temperature":

```

```
25, "Humidity": 30, "Light": 200}, "2": {"Location": "Bed room",
"Temperature": 18, "Humidity": 60, "Light": 150}, "3": {"Location":
"Garage", "Temperature": 30, "Humidity": 40, "Light": 70}}' to topic
'Enigma-HomeConditions'
```

```
Published message '{"0": {"Location": "Living room", "Temperature": 20,
"Humidity": 10, "Light": 250}, "1": {"Location": "Kitchen", "Temperature":
25, "Humidity": 30, "Light": 200}, "2": {"Location": "Bed room",
"Temperature": 18, "Humidity": 60, "Light": 150}, "3": {"Location":
"Garage", "Temperature": 30, "Humidity": 40, "Light": 70}}' to topic
'Enigma-HomeConditions'
```

 Test Client



Connection tcp://test.mosquitto.org:1883 ● ⏻

Subscriptions

The topic filter QoS 0 Subscribe

Subscribed topics

Enigma-HomeConditions QoS 0 ✕

Publish

The destination of the message QoS 0 ☐ Retain

The message text to be sent

Publish

Messages

2023-9-1 11:13:57.790 topic: Enigma-HomeConditions QoS 0

{"0": {"Location": "Living room", "Temperature": 20, "Humidity": 10, "Light": 250}, "1": {"Location": "Kitchen", "Temperature": 25, "Humidity": 30, "Light": 200}, "2": {"Location": "Bed room", "Temperature": 18, "Humidity": 60, "Light": 150}, "3": {"Location": "Garage", "Temperature": 30, "Humidity": 40, "Light": 70}}

2023-9-1 11:13:51.788 topic: Enigma-HomeConditions QoS 0

{"0": {"Location": "Living room", "Temperature": 20, "Humidity": 10, "Light": 250}, "1": {"Location": "Kitchen", "Temperature": 25, "Humidity": 30, "Light": 200}, "2": {"Location": "Bed room", "Temperature": 18, "Humidity": 60, "Light": 150}, "3": {"Location": "Garage", "Temperature": 30, "Humidity": 40, "Light": 70}}

Task 2:

Publish the contents of the worksheet as separate JSON objects for each row to a single topic in an MQTT broker.

```
import paho.mqtt.client as mqtt
import time
import json
import pandas as pd

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
    else:
        print("Connection failed with code {rc}")

# Create an MQTT client instance
client = mqtt.Client("PythonPub")

# Set the callback function
client.on_connect = on_connect

broker_address = "test.mosquitto.org" # broker's address
broker_port = 1883
keepalive = 5
qos = 0
publish_topic = "Enigma-HomeConditions"

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

# Start the MQTT loop to handle network traffic
client.loop_start()

# Publish loop
try:
    while True:
        # Publish a message to the send topic
        df = pd.read_excel('data.xlsx')
```

```

        json_data = df.to_dict(orient='records')
        for i,record in enumerate(json_data):
            print(record)
            data_out = json.dumps(record)
            client.publish(publish_topic,data_out)
            print(f"Published message '{data_out}' to topic
'{publish_topic}'\n")

        # Wait for a moment to simulate some client activity
        time.sleep(6)

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")

```

This Python code connects to a MQTT broker and feeds data from an Excel file to a specific MQTT topic. It accomplishes this by creating a MQTT client, defining a callback function for connection status, specifying the broker's details, and then entering a loop to publish data to the MQTT topic in JSON format, line wise. When finished, the program interrupts and disconnects from the broker. It essentially shows how to use Python to communicate data with other systems via MQTT.

Results:

```

Connected to MQTT broker

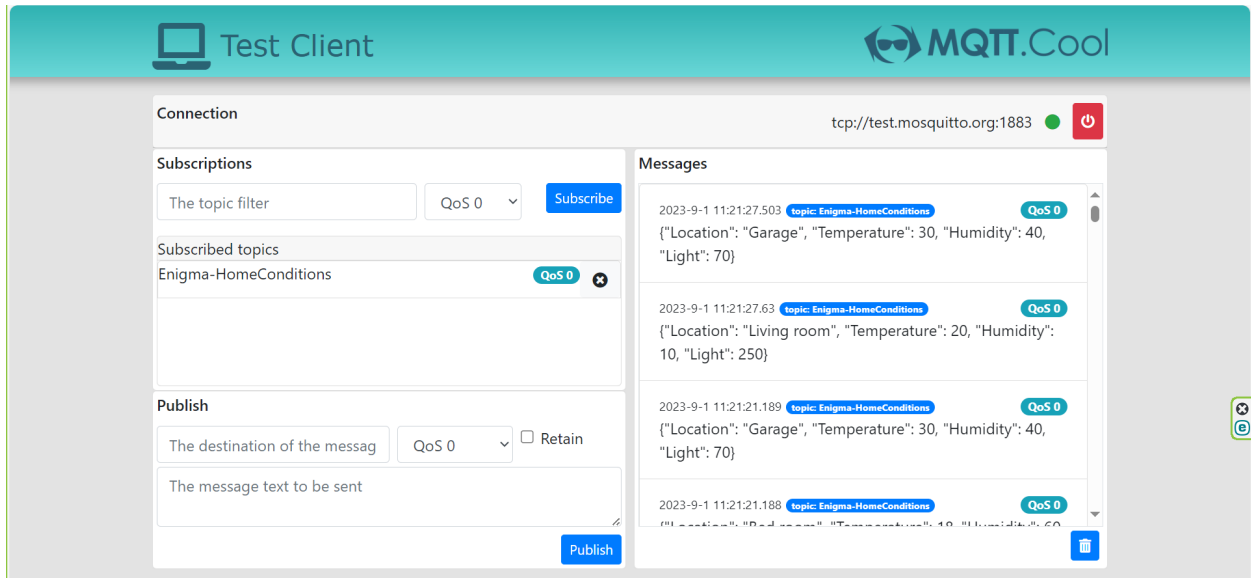
{'Location': 'Living room', 'Temperature': 20, 'Humidity': 10, 'Light':
250}
Published message '{"Location": "Living room", "Temperature": 20,
"Humidity": 10, "Light": 250}' to topic 'Enigma-HomeConditions'

{'Location': 'Kitchen', 'Temperature': 25, 'Humidity': 30, 'Light': 200}
Published message '{"Location": "Kitchen", "Temperature": 25, "Humidity":
30, "Light": 200}' to topic 'Enigma-HomeConditions'

{'Location': 'Bed room', 'Temperature': 18, 'Humidity': 60, 'Light': 150}
Published message '{"Location": "Bed room", "Temperature": 18, "Humidity":
60, "Light": 150}' to topic 'Enigma-HomeConditions'

```

```
{'Location': 'Garage', 'Temperature': 30, 'Humidity': 40, 'Light': 70}
Published message '{"Location": "Garage", "Temperature": 30, "Humidity": 40, "Light": 70}' to topic 'Enigma-HomeConditions'
```



Task 3:

Publish the contents of the worksheet as separate JSON objects for each row to different topics in an MQTT broker. The topic should be illustrative of the location.

```
import paho.mqtt.client as mqtt
import time
import json
import pandas as pd

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker\n")
    else:
        print("Connection failed with code {rc}")

# Create an MQTT client instance
```

```

client = mqtt.Client("PythonPub")

# Set the callback function
client.on_connect = on_connect

broker_address = "test.mosquitto.org" # broker's address
broker_port = 1883
keepalive = 5
qos = 0
#publish_topic = "Enigma-HomeConditions"

# Connect to the MQTT broker
client.connect(broker_address, broker_port, keepalive)

# Start the MQTT loop to handle network traffic
client.loop_start()

# Publish loop

try:
    while True:
        # Publish a message to the send topic
        df = pd.read_excel('data.xlsx')
        json_data = df.to_dict(orient='records')
        for i,record in enumerate(json_data):
            pub_topic = "Enigma-" + record["Location"]
            del record["Location"]
            data_out = json.dumps(record)
            client.publish(pub_topic,data_out)
            print(f"Published message '{data_out}' to topic
'{pub_topic}'\n")

            # Wait for a moment to simulate some client activity
            time.sleep(6)

except KeyboardInterrupt:
    # Disconnect from the MQTT broker
    pass
client.loop_stop()
client.disconnect()

print("Disconnected from the MQTT broker")

```

This Python code connects to a MQTT broker and sends data from an CSV file to several MQTT topics in real time (Namely Enigma-Living room, Enigma-Bed room, Enigma-Kitchen and Enigma-Garage), dynamically constructing topics based on the data's location. It defines a connection status callback function, specifies the MQTT broker's address and port, and initiates a MQTT loop for network communication. It reads data from the 'data.xlsx' file in a loop, converts each record to JSON format, then publishes it to individual MQTT topics named after the location of the data. After that, it waits 6 seconds before repeating the process. If the user interrupts it, it disconnects from the MQTT broker. This sample efficiently demonstrates how to use Python to transmit data with other systems over MQTT, establishing topics dynamically for different locations.

Results:

```
Connected to MQTT broker
```

```
Published message '{"Temperature": 20, "Humidity": 10, "Light": 250}' to  
topic 'Enigma-Living room'
```

```
Published message '{"Temperature": 25, "Humidity": 30, "Light": 200}' to  
topic 'Enigma-Kitchen'
```

```
Published message '{"Temperature": 18, "Humidity": 60, "Light": 150}' to  
topic 'Enigma-Bed room'
```

```
Published message '{"Temperature": 30, "Humidity": 40, "Light": 70}' to  
topic 'Enigma-Garage'
```

Test Client

MQTT.Cool

Connection

tcp://test.mosquitto.org:1883

Subscriptions

The topic filter

QoS 0

Subscribe

Subscribed topics

Enigma-Bed room

QoS 0

Publish

The destination of the message

QoS 0

☐ Retain

The message text to be sent

Publish

Messages

{"Temperature": 18, "Humidity": 60, "Light": 150}

2023-9-1 11:30:58.811

topic: Enigma-Bed room

QoS 0

{"Temperature": 78, "Humidity": 60, "Light": 150}

2023-9-1 11:30:52.659

topic: Enigma-Bed room

QoS 0

{"Temperature": 78, "Humidity": 60, "Light": 500}

2023-9-1 11:30:46.580

topic: Enigma-Bed room

QoS 0

{"Temperature": 18, "Humidity": 60, "Light": 500}

2023-9-1 11:30:40.457

topic: Enigma-Bed room

QoS 0

{"Temperature": 18, "Humidity": 60, "Light": 500}

Test Client

MQTT.Cool

Connection

tcp://test.mosquitto.org:1883

Subscriptions

The topic filter

QoS 0

Subscribe

Subscribed topics

Enigma-Living room

QoS 0

Publish

The destination of the message

QoS 0

☐ Retain

The message text to be sent

Publish

Messages

2023-9-1 11:26:49.464

topic: Enigma-Living room

QoS 0

{"Temperature": 20, "Humidity": 10, "Light": 250}

2023-9-1 11:26:43.341

topic: Enigma-Living room

QoS 0

{"Temperature": 20, "Humidity": 15, "Light": 250}

2023-9-1 11:26:37.223

topic: Enigma-Living room

QoS 0

{"Temperature": 20, "Humidity": 15, "Light": 250}

2023-9-1 11:26:31.182

topic: Enigma-Living room

QoS 0

{"Temperature": 20, "Humidity": 10, "Light": 250}

2023-9-1 11:26:25.101

topic: Enigma-Living room

QoS 0

Test Client

MQTT.Cool

Connection

tcp://test.mosquitto.org:1883

Subscriptions

The topic filter

QoS 0

Subscribe

Subscribed topics

Enigma-Kitchen

QoS 0

Publish

The destination of the message

QoS 0

☐ Retain

The message text to be sent

Publish

Messages

2023-9-1 11:32:17.896

topic: Enigma-Kitchen

QoS 0

{"Temperature": 25, "Humidity": 30, "Light": 200}

2023-9-1 11:32:11.856

topic: Enigma-Kitchen

QoS 0

{"Temperature": 25, "Humidity": 30, "Light": 200}

2023-9-1 11:32:5.771

topic: Enigma-Kitchen

QoS 0

{"Temperature": 25, "Humidity": 30, "Light": 200}

2023-9-1 11:31:59.776

topic: Enigma-Kitchen

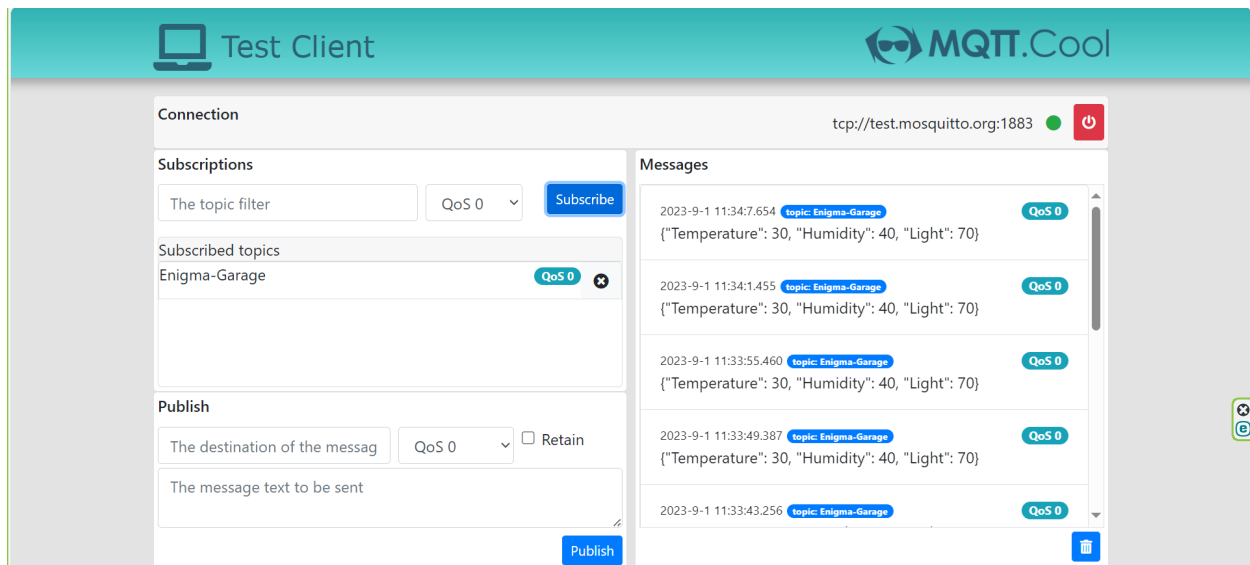
QoS 0

{"Temperature": 25, "Humidity": 30, "Light": 200}

2023-9-1 11:31:53.696

topic: Enigma-Kitchen

QoS 0



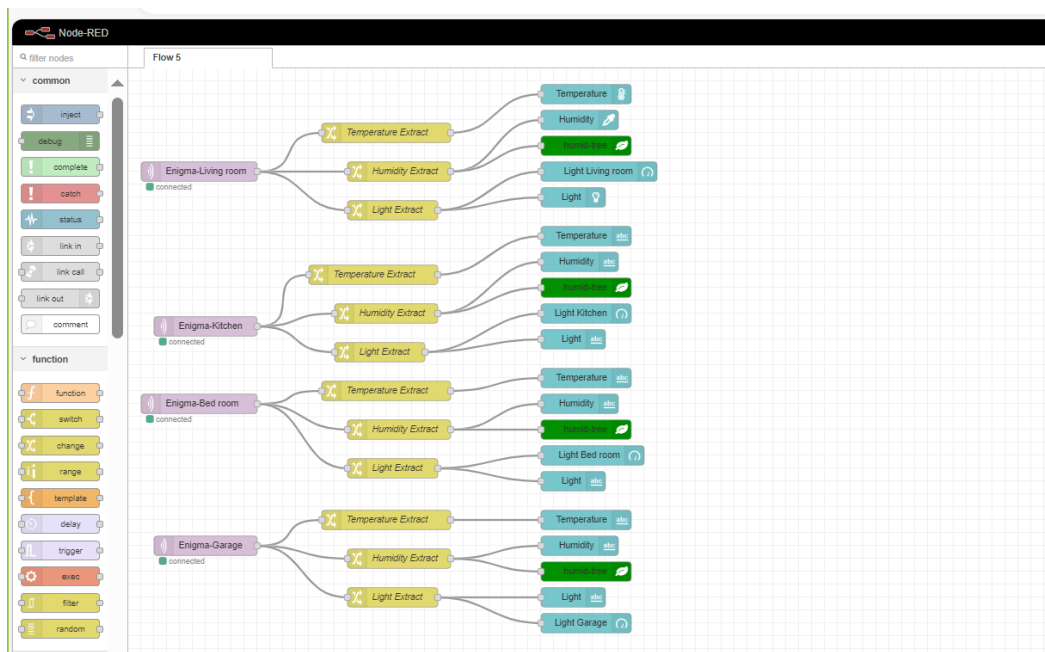
All the above codes can adhere to value changes as well.

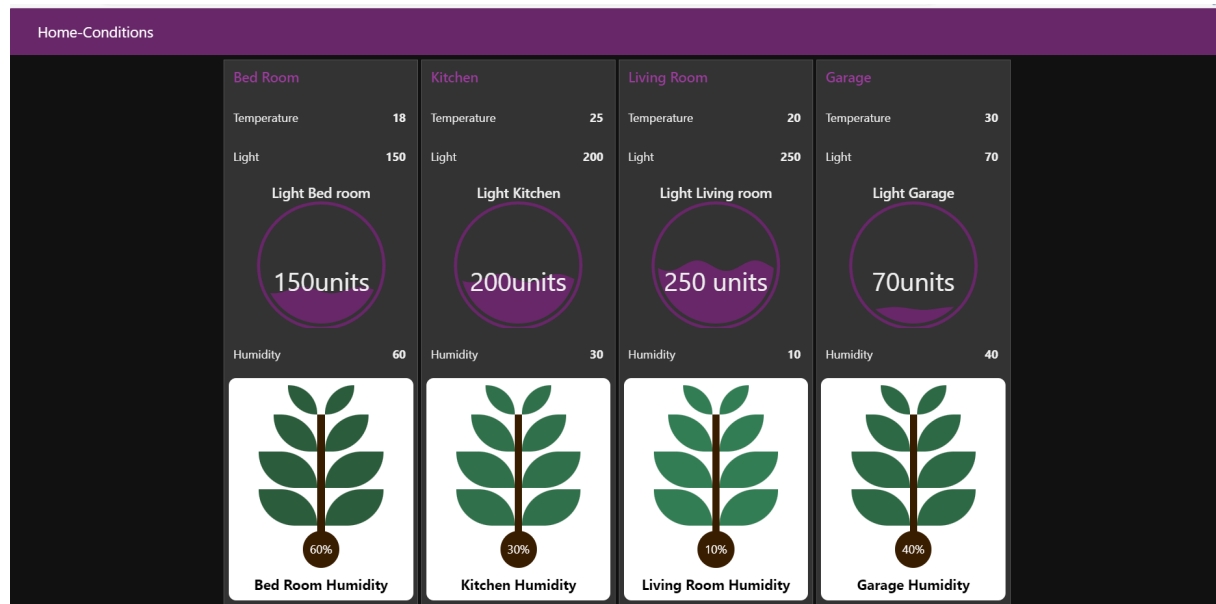
Ex: While the program was running we changed the value of the Humidity living room from 10 to 15, and the relevant data was transmitted accurately.

While the program was running we changed the value of the Light bedroom from 150 to 500, and the Temperature from 18 to 78, the relevant data was transmitted accurately.

Task 4:

NodeRed Dashboard

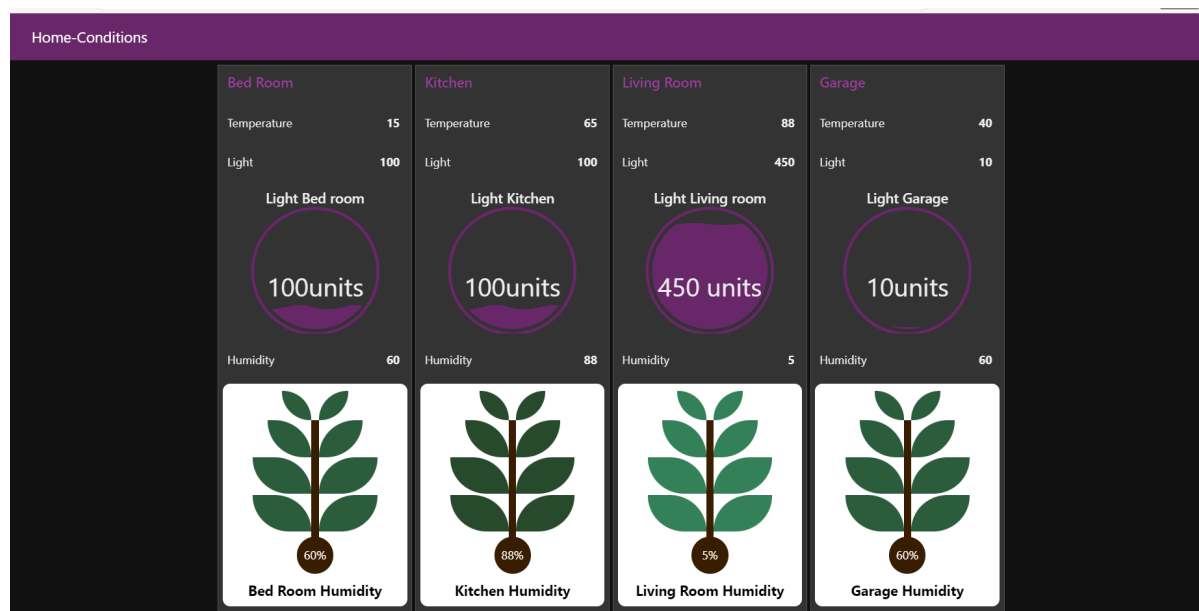




Then we changed the values as in the CSV file below.

	A	B	C	D	E	F
1	Location	Temperature	Humidity	Light		
2	Living room	88	5	450		
3	Kitchen	65	88	100		
4	Bed room	15	60	100		
5	Garage	40	60	10		
6						

The values changed accordingly.



Accordingly, we can conclude that the system pertains to any variations to the original data file.