

**Department of Electronic & Telecommunication
Engineering
University of Moratuwa
EN3251 Internet of Things**



**Laboratory Exercise 2
Information transfer with MQTT and HTTP using JSON**

KURUPPU M.P.	210325M
PEIRIS D.L.C.J.	210454G
PERERA L.C.S.	210463H

Date - 2024.09.13

Contents

1	Step 01	2
1.1	Introduction for Json	2
1.2	Key Features of JSON	2
1.3	Benefits of JSON in IoT	2
1.4	Uses of JSON in IoT	3
2	Step 02	4
3	Step 03	5
3.1	Publisher Code	5
3.2	Subscriber Code	8
3.3	Discussion	10
4	Step 04	11
4.1	Open Weather Code	11
5	Homework	13
5.1	Using Publisher Message for Node-Red	13
5.2	Direct Implementation within Node-Red	16

1 Step 01

1.1 Introduction for Json

JSON (JavaScript Object Notation) is a lightweight, text-based data interchange format that is easy for both humans and machines to read and write. It's widely used for data exchange in various applications, including the Internet of Things (IoT).

1.2 Key Features of JSON

- Text-based Format: It's primarily used to represent structured data as text.
- Human-readable: Data is easy to understand and manually create.
- Language-independent: Although it originated from JavaScript, it can be used with many programming languages.
- Lightweight: It has minimal overhead, making it efficient in terms of network traffic and processing.

1.3 Benefits of JSON in IoT

1. Readability: JSON's clear structure makes it easy to read and understand. This is beneficial for debugging and data interpretation.
2. Lightweight: JSON's simple format minimizes the amount of data being transferred, which is crucial for IoT devices with limited bandwidth and processing power.
3. Compatibility: JSON is supported by many programming languages, making it a versatile choice for data interchange across different systems.
4. Ease of Parsing: Most programming languages offer libraries or built-in functions to parse JSON data efficiently, simplifying data handling in IoT applications.

1.4 Uses of JSON in IoT

1. Data Exchange Between IoT Devices:

IoT devices often need to transmit data from sensors to a central server or cloud platform. JSON is commonly used to encode sensor readings, such as temperature, humidity, and GPS coordinates, for easy transmission.

Example:

```
{
  "sensor_id": "A1B2C3",
  "temperature": 22.5,
  "humidity": 60,
  "location": {
    "latitude": 51.5074,
    "longitude": 0.1278
  }
}
```

2. Configuration Management:

IoT devices require configuration parameters (e.g., network settings, device behavior, thresholds for sensors). JSON is used to send configuration data from the server to the IoT device or to retrieve settings from the device.

3. Control Commands:

JSON is used to send control messages (e.g., turn on/off, set a parameter) to IoT devices from a central system or mobile application. Devices can interpret JSON messages to perform actions.

Example:

```
{
  "command": "turn_on",
  "device": "light",
  "brightness": 80
}
```

4. MQTT with JSON Payloads:

JSON is often used as the message payload in MQTT (Message Queuing Telemetry Transport) messages. Devices publish/subscribe to topics with JSON-formatted data, making communication efficient and structured.

Example MQTT payload:

```
{
  "topic": "home/living_room/temperature",
  "temperature": 21.0,
  "unit": "C"
}
```

5. Data Storage and Logging:

JSON is often used to store data generated by IoT devices. For example, historical

sensor data can be stored in JSON format in databases, making it easy to analyze later.

Example:

```
[
  { "timestamp": "2024-09-12T08:00:00Z", "temperature": 21.0 },
  { "timestamp": "2024-09-12T09:00:00Z", "temperature": 22.5 }
]
```

6. IoT Dashboard Updates:

JSON is used to update dashboards with real-time data from IoT devices. JSON-formatted data streams from the devices to the dashboard, where it is displayed to users.

Example:

```
{
  "dashboard": "SmartHome",
  "data": {
    "temperature": 22,
    "humidity": 45,
    "status": "online"
  }
}
```

2 Step 02

Done

3 Step 03

3.1 Publisher Code

The below code represents the MQTT publisher which reads an object containing multiple string-value pairs from a file called data, and publish it to a Broker on Topic of IOT_JSON

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
import json
read_file_name = "data.json"

# Function to read data from a JSON file
def read_json_file(file_path):
    with open(read_file_name) as json_file:
        file_data= json.load(json_file)
    data_out=json.dumps(file_data) #encode object to JSON
    return data_out

# 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(mqtt_client.CallbackAPIVersion.VERSION1, "PythonPub"
    )

# Set the callback function
client.on_connect = on_connect

broker_address = "test.mosquitto.org" # broker's address
broker_port = 1883
keepalive = 60
qos = 2
publish_topic = "IOT_JSON"

# 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:
```

Lab Exercise 2

```
# Publish a message to the send topic

value = read_json_file(read_file_name )
client.publish(publish_topic,value, qos=qos)
print(f"Published message '{value}' 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")
```

Wireshark Packet Capture

No.	Time	Source	Destination	Protocol	Length	Info
73550	433.955035	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	2001:41d0:1:925e::1	MQTT	97	Connect Command
73580	434.160656	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	2001:41d0:1:925e::1	MQTT	872	Publish Message (id=1) [IOT_JSON]
73581	434.161800	2001:41d0:1:925e::1	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	MQTT	78	Connect Ack
73604	434.364611	2001:41d0:1:925e::1	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	MQTT	78	Publish Received (id=1)
73605	434.364844	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	2001:41d0:1:925e::1	MQTT	78	Publish Release (id=1)
73638	434.591671	2001:41d0:1:925e::1	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	MQTT	78	Publish Complete (id=1)

Figure 1: Packet Capturing at Publisher

Observations

73580	434.160656	2402:4000:21c4:136b:cd21:93f0:7d05:6c2c	2001:41d0:1:925e::1	MQTT	872	Publish Message (id=1) [IOT_JSON]	
Frame 73580: 872 bytes on wire (6976 bits), 872 bytes captured (6976 bits) on interface \Device\NPF{...}							
Ethernet II, Src: Intel_a3:3b:ad (f4:de:e3:a3:3b:ad), Dst: GuangzhouToz_18:4a:12 (98:a9:42:18:4a:12)							
Internet Protocol Version 6, Src: 2402:4000:21c4:136b:cd21:93f0:7d05:6c2c, Dst: 2001:41d0:1:925e::1							
Transmission Control Protocol, Src Port: 62958, Dst Port: 1883, Seq: 24, Ack: 1, Len: 798							
MQ Telemetry Transport Protocol, Publish Message							
Header Flags: 0x34, Message Type: Publish Message, QoS Level: Exactly once delivery (Assured)							
Msg Len: 795							
Topic Length: 8							
Topic: IOT_JSON							
Message Identifier: 1							
Message [-]: 7b22696e7669726f6e6d56e74223a207b2274656d7065726174757265523a207b22637572726556							
0000	00	a9	42	18	4a	12 f4 4e e3 a3 3b ad 86 dd 60 0b	OT_JSON: {"envir
0010	92	ee	03	32	06	40 24 02 40 00 21 c4 13 6b cd 21	onment": {"tempe
0020	93	f0	7d	05	6c	2c 20 01 41 d0 00 01 92 5e 00 00	ature": {"curre
0030	00	00	00	00	00	01 f5 ee 07 5b 15 74 ef e8 95 95	nt": 25.0, "unit
0040	eb	47	50	18	02	02 da df 00 00 34 9b 06 00 08 49	": "Celsius", "h
0050	4f	54	5f	4a	53	4f 4e 00 01 7b 22 65 6e 76 69 72	igh": 30.0, "low
0060	6f	6e	6d	65	6e	74 22 3a 20 7b 22 74 65 6d 78 65	": 20.0, "timest
0070	72	61	74	75	72	65 22 3a 20 7b 22 63 75 72 72 65	amp": "2 024-09-1
0080	6e	74	22	3a	20	32 35 2e 30 2c 20 22 75 6e 69 74	3T14:30: 00Z", "
0090	22	3a	20	22	43	65 6c 73 69 75 73 22 2c 20 22 68	humidity": {"curre
00a0	69	67	68	22	3a	20 33 30 2e 30 2c 20 22 6c 6f 77	rent": 6.0, "unit
00b0	22	3a	20	32	30	2e 30 2c 20 22 74 69 6d 65 73 74	": "%", "high":
00c0	61	6d	70	22	3a	20 22 32 30 32 34 2d 30 39 2d 31	80, "low": 40, "
00d0	33	54	31	34	3a	33 30 3a 30 30 5a 22 7d 2c 20 22	timestamp": "202
00e0	68	75	6d	69	64	69 74 79 22 3a 20 7b 22 63 75 72	4-09-13T 14:30:00
00f0	72	65	6e	74	22	3a 20 36 30 2c 20 22 75 6e 69 74	Z"}], "weather":
0100	22	3a	20	22	25	22 2c 20 22 68 69 67 68 22 3a 20	["description":
0110	38	30	2c	20	22	6c 6f 77 22 3a 20 34 30 2c 20 22	"Partly cloudy"
0120	74	69	6d	65	73	74 61 6d 70 22 3a 20 22 32 30 32	, "forecast": [{
0130	34	2d	30	39	2d	31 33 54 31 34 3a 33 30 3a 30 30	"date": "2024-09
0140	5a	22	7d	7d	2c	20 22 77 65 61 74 68 65 72 22 3a	-14", "high_temp
0150	20	7b	22	64	65	73 63 72 69 70 74 69 6f 6e 22 3a	erature": 28.0,
0160	20	22	50	61	72	74 6c 79 20 63 6c 6f 75 64 75 22	"low_tem perature
0170	2c	20	22	66	6f	72 65 63 61 73 74 22 3a 20 5b 7b	": 22.0, "precip
0180	22	64	61	74	65	22 3a 20 22 32 30 32 34 2d 30 39	itation_chance":
0190	2d	31	34	22	2c	20 22 68 69 67 68 5f 74 65 6d 70	10), ("date": "
01a0	65	72	61	74	75	72 65 22 3a 20 32 38 2e 30 2c 20	2024-09-15", "hi
01b0	22	6c	6f	77	5f	74 65 6d 70 65 72 61 74 75 72 65	gh_tempera ture":
01c0	22	3a	20	32	32	2e 30 2c 20 22 70 72 65 63 69 70	26.0, " low_tem
01d0	69	74	61	74	69	6f 6e 5f 63 68 61 6e 63 65 22 3a	erature": 21.0,
01e0	20	31	30	7d	2c	20 7b 22 64 61 74 65 22 3a 20 22	"precipitation_c
01f0	32	30	32	34	2d	30 39 2d 31 35 22 2c 20 22 68 69	itation_chance":
0200	67	68	5f	74	65	6d 70 65 72 61 74 75 72 65 22 3a	10), ("date": "
0210	20	32	36	2e	30	2c 20 22 6c 6f 77 5f 74 65 6d 70	2024-09-15", "hi
0220	65	72	61	74	75	72 65 22 3a 20 32 31 2e 30 2c 20	gh_tempera ture":
0230	22	70	72	65	63	69 70 69 74 61 74 69 6f 6e 5f 63	26.0, " low_tem

Figure 2: Payload of the Published Message

1. Payload Format: The payload is successfully transmitted as a JSON object. The IOT_JSON topic contains structured data with key-value pairs, making it useful for

IoT monitoring or control applications.

2. **Correct Functionality:** From the packet capture, it seems your MQTT client is functioning as expected—reading the JSON object, publishing it to the topic, and ensuring that it reaches the broker with no packet errors.

3.2 Subscriber Code

The below code represents the MQTT subscriber which receives an object containing multiple string-value pairs that we have published and writes it to a file called `data_received.json`.

```
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time
write_file_name = "data_received.json"
import json

# Callback when the client connects to the MQTT broker
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT broker")
        client.subscribe(subscribe_topic, qos) # Subscribe to the
            receive topic
    else:
        print("Connection failed with code {rc}")

# Callback when a message is received from the subscribed topic
def on_message(client, userdata, msg):
    print ("Message received " + "on " + subscribe_topic + ": " + str(
        msg.payload.decode("utf-8")))
    value = str(msg.payload.decode("utf-8"))
    with open(write_file_name, 'w') as json_file:
        json.dump(value, json_file, indent=4) # The 'indent' parameter
            adds pretty formatting
    print("Data has been written to", write_file_name)

# Create an MQTT client instance
client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "PythonSub"
    )

# Set the callback functions
client.on_connect = on_connect
client.on_message = on_message

# Connect to the MQTT broker
broker_address = "test.mosquitto.org" # broker's address
broker_port = 1883
keepalive = 60
qos = 2

subscribe_topic = input ('Enter the topic to subscribe to: ')
client.connect(broker_address, broker_port, keepalive)

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

# Subscribe loop
```

Lab Exercise 2

```
try:
    while True:
        time.sleep(6)

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

print("Disconnected from the MQTT broker")
```

Wireshark Packet Capture

1899	22:34:45.114518	2402::d000:8138:a22:66:9d5:c1e1:d364	2001::41d0:1:925e::1	MQTT	97	Connect Command
1906	22:34:45.722340	2001::41d0:1:925e::1	2402::d000:8138:a22:66:9d5:c1e1:d364	MQTT	70	Connect Ack
1907	22:34:45.722235	2402::d000:8138:a22:66:9d5:c1e1:d364	2001::41d0:1:925e::1	MQTT	89	Subscribe Request (id=1) [IOT_350N]
1909	22:34:45.992448	2001::41d0:1:925e::1	2402::d000:8138:a22:66:9d5:c1e1:d364	MQTT	79	Subscribe Ack (id=1)
3147	22:34:56.539515	2001::41d0:1:925e::1	2402::d000:8138:a22:66:9d5:c1e1:d364	MQTT	872	Publish Message (id=1) [IOT_350N]
3148	22:34:56.540072	2402::d000:8138:a22:66:9d5:c1e1:d364	2001::41d0:1:925e::1	MQTT	70	Publish Received (id=1)
3188	22:34:56.809342	2001::41d0:1:925e::1	2402::d000:8138:a22:66:9d5:c1e1:d364	MQTT	78	Publish Release (id=1)
3190	22:34:56.808960	2402::d000:8138:a22:66:9d5:c1e1:d364	2001::41d0:1:925e::1	MQTT	78	Publish Complete (id=1)
3443	22:34:58.829275	2402::d000:8138:a22:66:9d5:c1e1:d364	2001::41d0:1:925e::1	MQTT	76	Disconnect Req

Figure 3: Packet Capturing at Subscriber

Observations

3147	22:34:56.539515	2001::41d0:1:925e::1	2402::d000:8138:a22:66:9d5:c1e1:d364	MQTT	872	Publish Message (id=1) [IOT_350N]
Frame 3147: 872 bytes on wire (6976 bits), 872 bytes captured (6976 bits) on interface \Device\NPF... [5C6d...]						
Ethernet II, Src: HuaweiTechno... [44:8f:3e:1c:20:db:4d:8f:3e], Dst: Intel_3e:dd:57 (e8:84:a5:3e:dd:57)						
Internet Protocol Version 6, Src: 2001::41d0:1:925e::1, Dst: 2402::d000:8138:a22:66:9d5:c1e1:d364						
Transmission Control Protocol, Src Port: 1883, Dst Port: 53916, Seq: 10, Ack: 39, Len: 798						
MQ Telemetry Transport Protocol, Publish Message						
Header Flags: 0x03, Message Type: Publish Message, QoS Level: Exactly once delivery (Assured Delivery)						
Msg Len: 795						
Topic: IOT_350N						
Message Identifier: 1						
Message [..]: 7b22656e766972666e6d656e74223a207b2274656d7065726174757265223a207b2263757272656e74223a203						
0000	e8	84	a5	3e	dd	57
0010	58	6d	03	32	06	2e
0020	00	00	00	00	01	24
0030	09	d5	c1	e1	d3	64
0040	31	74	50	18	01	f9
0050	4f	54	5f	4a	53	4f
0060	6f	6e	6d	65	6e	74
0070	72	61	74	75	72	65
0080	69	67	68	22	3a	20
0090	22	3a	20	22	43	65
00a0	33	54	31	34	3a	33
00b0	68	75	6d	69	64	69
00c0	61	6d	70	22	3a	20
00d0	5a	22	70	78	2e	20
00e0	20	7b	22	64	65	73
00f0	20	22	50	61	72	74
0100	22	3a	20	22	25	2c
0110	38	30	2c	20	22	6c
0120	74	69	6d	65	73	74
0130	34	2d	30	39	2d	31
0140	5a	22	70	78	2e	20
0150	20	7b	22	64	65	73
0160	20	22	50	61	72	74
0170	2c	20	22	66	6f	72
0180	22	64	61	74	65	22
0190	2d	31	34	22	2c	20
01a0	65	72	61	74	75	72
01b0	22	3a	20	32	3e	2a
01c0	22	3a	20	32	3e	2a
01d0	69	74	61	74	69	6f
01e0	20	31	30	78	2e	20
01f0	32	30	32	34	2d	30
0200	67	68	5f	74	65	6d
0210	20	32	36	2e	30	2c
0220	65	72	61	74	75	72
0230	22	70	72	65	63	69
0240	68	61	6e	63	65	22
0250	6c	6f	63	61	74	69
0260	79	22	3a	20	22	43
0270	63	6f	75	6e	74	72
0280	61	6e	6b	61	22	2c
0290	74	65	73	22	3a	20

Figure 4: Payload Received at Subscriber

1. The packet flow follows the typical sequence for MQTT with QoS 2 (Exactly Once Delivery). Wireshark confirms the MQTT operations such as Connect, Publish, Publish Received (PUBREC), Publish Release (PUBREL), Publish Complete (PUBCOMP), and the receipt of messages. This ensures that the message is delivered exactly once, preventing duplicates.

2. The payload being transmitted is structured as a JSON object. Wireshark successfully captured and decoded the payload, showing its contents clearly in the hex view.
3. Since the payload matches the expected JSON structure, we can confirm that the data integrity is preserved across the MQTT communication, even with the additional steps involved in QoS 2 to ensure reliable message delivery.

3.3 Discussion

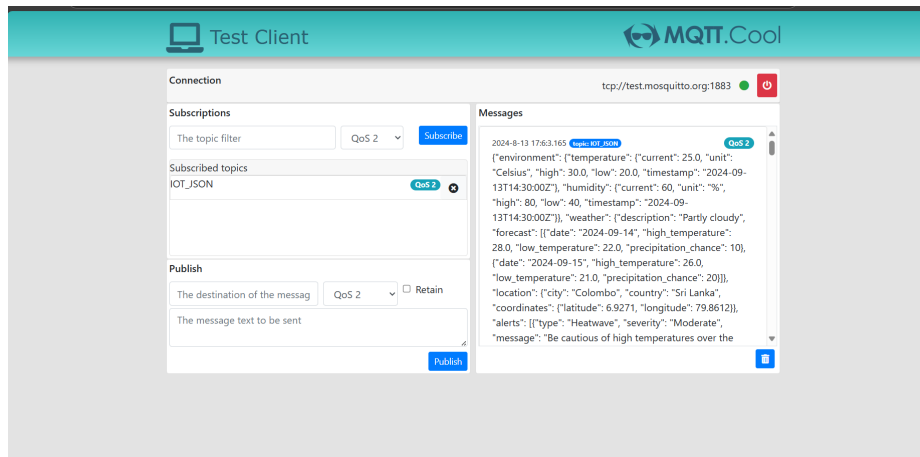


Figure 5: Test Client

The above tasks demonstrated the successful use of MQTT for publishing and subscribing to JSON data. The JSON format's flexibility and readability made it an ideal choice for transmitting structured sensor data. The use of Wireshark confirmed that the MQTT messages, including JSON payloads, were correctly transmitted and received. Both publishing and subscribing processes successfully handled the JSON format, showing its suitability for transmitting complex data structures in IoT scenarios. (Objects containing multiple string-value pairs)

4 Step 04

4.1 Open Weather Code

```
import requests
import json

write_file_name = "openweathermap_data.json"

def get_weather(city_name, api_key):
    base_url = "http://api.openweathermap.org/data/2.5/weather"

    # Parameters for the API request
    params = {
        'q': city_name,          # City name
        'appid': api_key,        # API key
        'units': 'metric'        # Units for temperature (metric,
                                # imperial, or standard)
    }

    # Make the HTTP GET request to the OpenWeather API
    response = requests.get(base_url, params=params)

    # Check if the request was successful
    if response.status_code == 200:
        # Parse JSON data from the response
        data = response.json()

        # Write the JSON data to a file
        with open(write_file_name, 'w') as json_file:
            json.dump(data, json_file, indent=4) # The 'indent'
            # parameter adds pretty formatting

        print("Data has been written to", write_file_name)

        # Extract relevant information from the response
        main = data['main']
        weather = data['weather'][0]
        wind = data['wind']

        # Print weather information
        print(f"Weather in {city_name.capitalize()}:")
        print(f"Temperature: {main['temp']} C ")
        print(f"Humidity: {main['humidity']}%")
        print(f"Condition: {weather['description'].capitalize()}")
        print(f"Wind Speed: {wind['speed']} m/s")
    else:
        # Print error message if request was unsuccessful
        print(f"Error: Could not retrieve weather for '{city_name}' (
            HTTP {response.status_code})")
        print(f"Response Text: {response.text}")

if __name__ == "__main__":
```

Lab Exercise 2

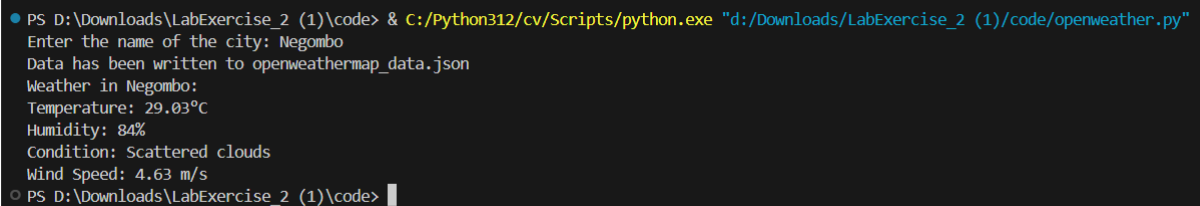
```
# Your OpenWeather API key
api_key = "17c3824f1382e7309370c91d8173294c"

# Get city name from user
city_name = input("Enter the name of the city: ")

# Fetch and display weather information
get_weather(city_name, api_key)
```

The above Python code fetches current weather data for a user-specified city using the OpenWeather API. It saves the data in a JSON file and prints the weather details, including temperature, humidity, weather conditions, and wind speed. If the API request fails, it displays an error message.

The results obtained:



```
PS D:\Downloads\LabExercise_2 (1)\code> & C:/Python312/cv/Scripts/python.exe "d:/Downloads/LabExercise_2 (1)/code/openweather.py"
Enter the name of the city: Negombo
Data has been written to openweathermap_data.json
Weather in Negombo:
Temperature: 29.03°C
Humidity: 84%
Condition: Scattered clouds
Wind Speed: 4.63 m/s
PS D:\Downloads\LabExercise_2 (1)\code> |
```

Figure 6: Openweather Results

We have written a json file containing the obtained results as follows:

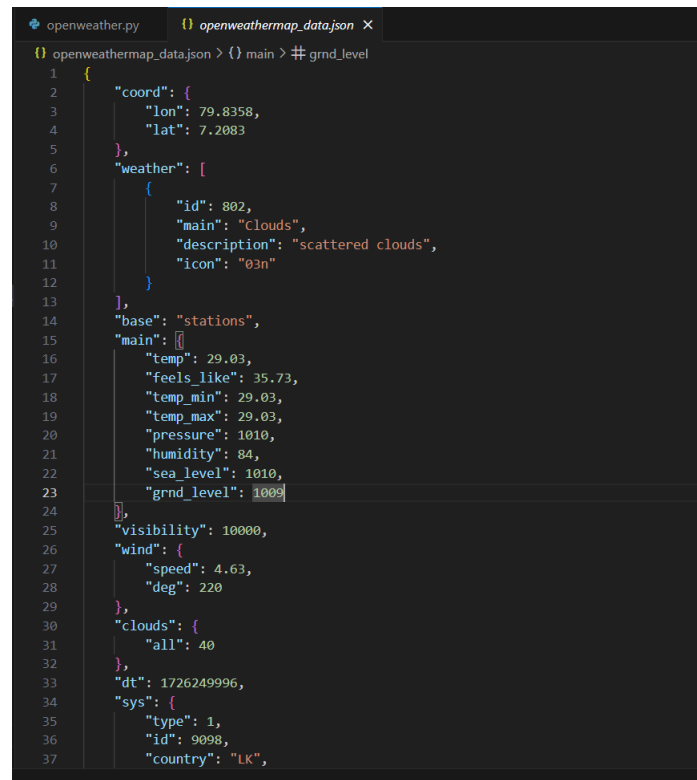


Figure 7: Generated Json file data obtained by Open Weather

5 Homework

5.1 Using Publisher Message for Node-Red

By using the following python code we can obtained the weather data from <https://openweathermap.org> and publish the message to MQTT In in the Red-Note.

```

import requests
import json
from paho.mqtt import client as mqtt_client
import paho.mqtt.client as mqtt
import time

def get_weather(city_name, api_key):
    base_url = "http://api.openweathermap.org/data/2.5/weather"

    # Parameters for the API request
    params = {
        'q': city_name,           # City name
        'appid': api_key,         # API key
        'units': 'metric'         # Units for temperature (metric, imperial
                                # , or standard)
    }

```

```
# Make the HTTP GET request to the OpenWeather API
response = requests.get(base_url, params=params)

print(f"Response Status Code: {response.status_code}")

# Check if the request was successful
if response.status_code == 200:
    # Parse JSON data from the response
    data = response.json()

    # Extract relevant information from the response
    main = data['main']
    weather = data['weather'][0]
    wind = data['wind']

    # Create a dictionary with weather data for the city
    weather_data = {
        'city': city_name.capitalize(),
        'temperature': main['temp'],
        'humidity': main['humidity'],
        'condition': weather['description'].capitalize(),
        'wind_speed': wind['speed']
    }

    return weather_data
else:
    # Return an error message if the request was unsuccessful
    print(f"Error: Could not retrieve weather for '{city_name}' (
        HTTP {response.status_code})")
    return None

def main():
    #OpenWeather API key
    api_key = "79d08796032858d8504f2f893c77115f"

    # Get weather data for 3 cities
    cities = ["Ratnapura", "Nugegoda", "Negombo"]

    # Initialize an empty list to hold weather data for all cities
    weather_data_list = []

    for city in cities:
        city_weather = get_weather(city, api_key)
        if city_weather:
            weather_data_list.append(city_weather)

    # Store weather data directly
    weather_data_json = json.dumps(weather_data_list, indent=4)

    # MQTT client setup
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT broker\n")
        else:
```

```
        print(f"Connection failed with code {rc}")

client = mqtt.Client(mqtt_client.CallbackAPIVersion.VERSION1, "
    PythonPub")
client.on_connect = on_connect

broker_address = "test.mosquitto.org" # Broker's address
broker_port = 1883
keepalive = 60
qos = 2
publish_topic = "IOT_JSON"

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

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

try:
    while True:
        # Publish the weather data directly
        client.publish(publish_topic, weather_data_json, qos=qos)
        print(f"Published message '{weather_data_json}' 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")

if __name__ == "__main__":
    main()
```


5.2 Direct Implementation within Node-Red

Node-Red Flow Diagram

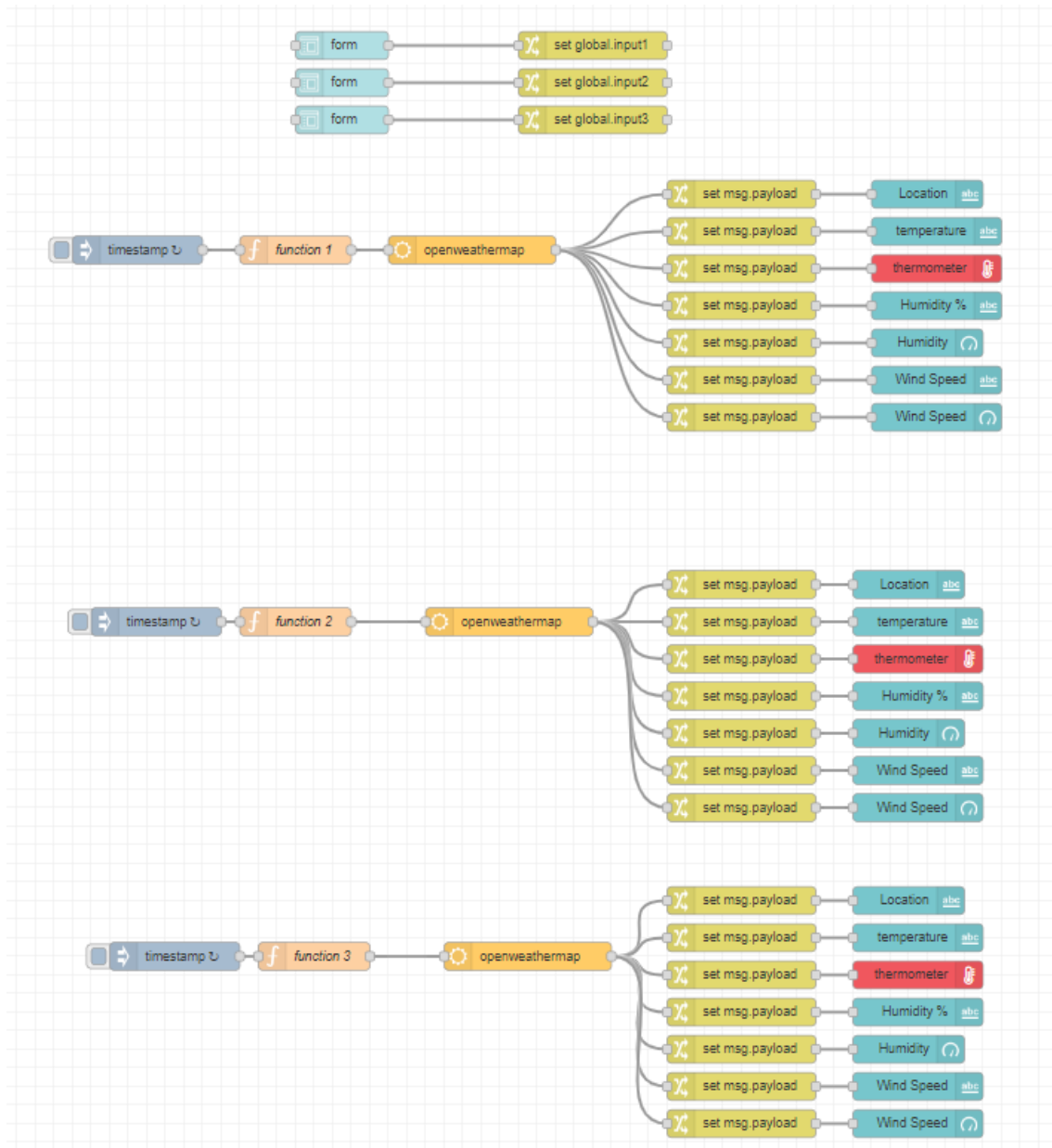


Figure 8: Node-Red Flow

Dash Board

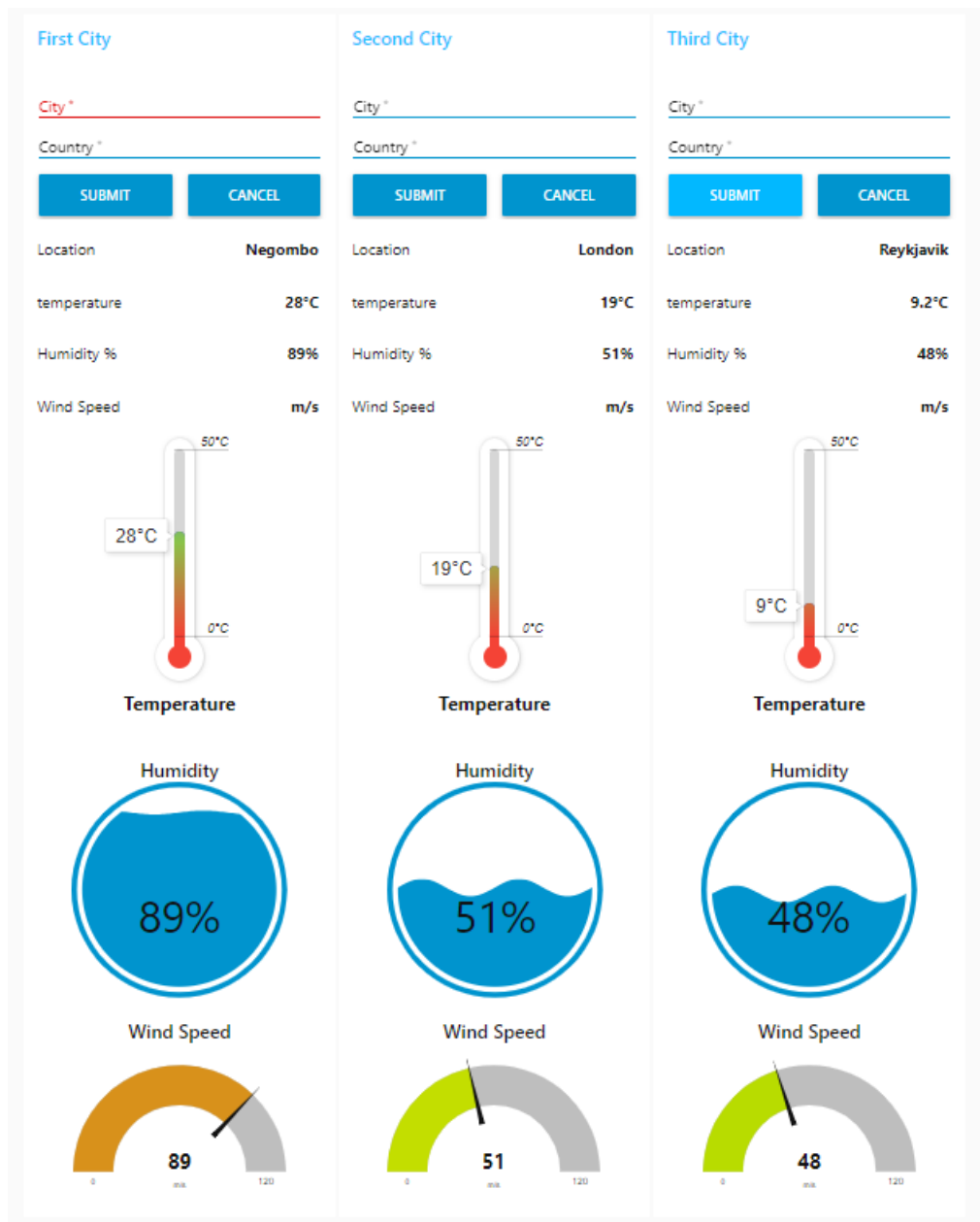


Figure 9: Dash Board

Implementation Summery

Our Node-RED dashboard integrated with the OpenWeatherMap API, designed to display real-time weather data for multiple cities. The dashboard provides a clear, graphical representation of temperature, humidity, and wind speed, utilizing interactive elements such as thermometers, humidity gauges, and wind speed dials. The underlying Node-RED flow connects input forms to the API, where user-specified city data is processed. Timestamp nodes initiate API requests, and function nodes handle the retrieval and formatting of weather data, which is then routed to various output widgets on the dashboard. This setup allows for dynamic weather monitoring, offering users an intuitive interface for accessing environmental conditions from the OpenWeatherMap service.