Weather Station Project

SWE30011 IOT PROGRAMMING - GROUP ASSIGNMENT

TUTOR: SAMUAL GOLDING

TUESDAY 10:30AM TO 12:30PM

TEAM MEMBERS:

GURSHAN DHALIWAL (102578347)
LE BAO DUY NGUYEN (102449993)
BUNDY HUANG (103302958)
VIRUL VINWATH (102625159)

TABLE OF CONTENTS

WEATHER STATION PROJECT

INTRODUCTION

TOPIC BACKGROUND

With the final project for SWE30011 consisting of illustrating how group members can come together to develop a project which demonstrates the groups knowledge on Arduinos, RPi and cloud communication that they have acquired from tutorials hence why the group has decided that they would like to replicate a weather station. Whilst this project has been explored already and is far from unique it is deemed to be the ideal project which not only tests group members but also is in line with exhibiting practical knowledge of sensors/actuators, edge devices, IoT communication, IoT cloud computing, APIs, or web servers, IoT programming, data collection and analysis.

An IoT system consists of:

- Sensor (Publisher): A device or module that detects the surroundings and sends the information to the server.
- Actuator (Subscriber): Make action based on given information. Usually a fan, buzzer, etc.
- Data: Data from the sensor are stored within the cloud (i.e. ThingsBoard) and used for processing, analysing, and monitoring.
- Server: Data is transferred to server for hosting and processing. Server transmits data to the subscriber through MQTT protocol and fetches data using ThingsBoard Gateway API.
- Web Interface: HTML/JavaScript are used to fetch and display data from ThingsBoard's latest telemetry via WebSocket API.

PROPOSED SYSTEM

The project is inspired by the real-world use case of a smoke/temperature detector. When temperature reaches above/below a set threshold, the user will be notified by the buzzer/fan.

This project is using multiple Arduino boards, a DHT11 humidity sensor and a temperature potentiometer to record surrounding temperature/humidity and transfer through to ThingsBoard using serial communication and MQTT protocol. The data will then be displayed using ThingsBoard in the form of an analytical dashboard. Data is fetched from ThingsBoard via Python and transferred back for Arduino to take action accordingly.

Virtual Raspberry Pi is a Linux Debian 64-bit environment. 4 edge devices represent 4 different virtual machines, 1 for humidity, 1 for temperature, 1 for fan and 1 for buzzer. 1 extra edge server is created to act as a cloud server with its serial port turned off.

Using ThingsBoard, 4 edge devices are connected and communicate with each other to 1 cloud edge server. 2 edge devices publish temperature and humidity, whereas 2 others are the fan and buzzer which respond accordingly to the data fetched from ThingsBoard. Bidirectional communication is established by manually uploading data to ThingsBoard using command prompt query.

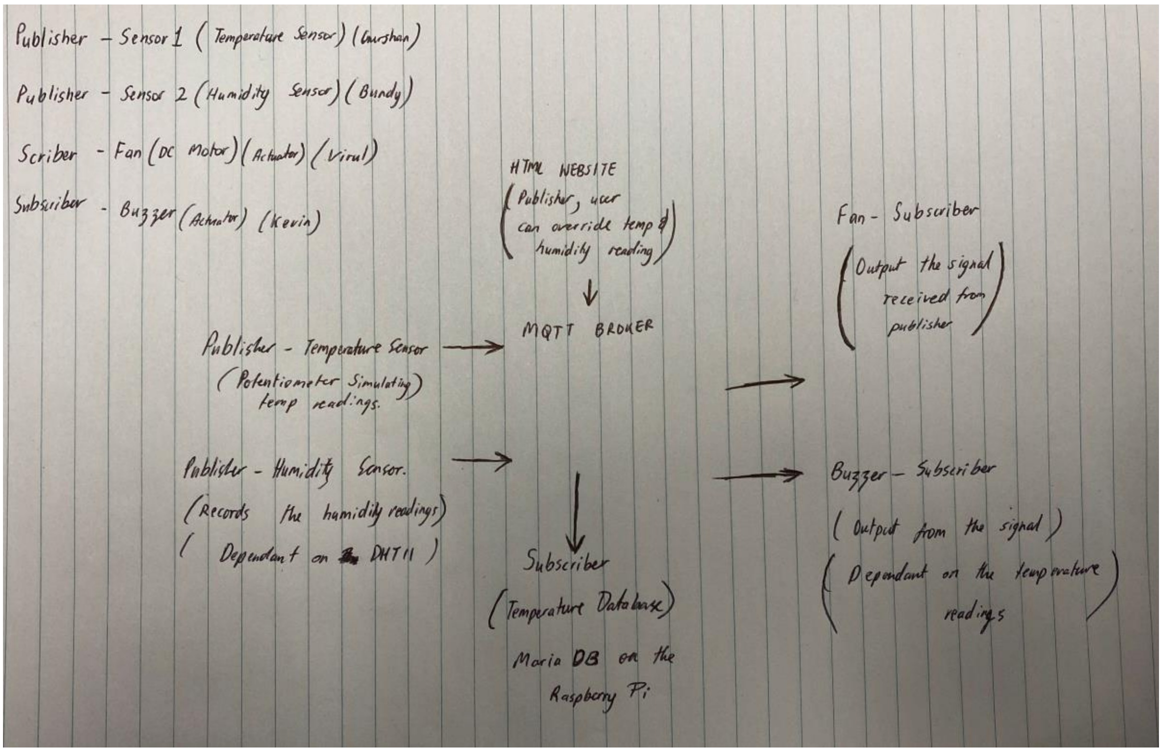
```
mosquitto_pub -d -h "172.20.10.7" -t "v1/devices/me/attributes" -u "TEMP" -m "{"temperature": 20}"
mosquitto_pub -d -h "172.20.10.7" -t "v1/devices/me/attributes" -u "HUMI" -m "{"humidity": 50}"
```

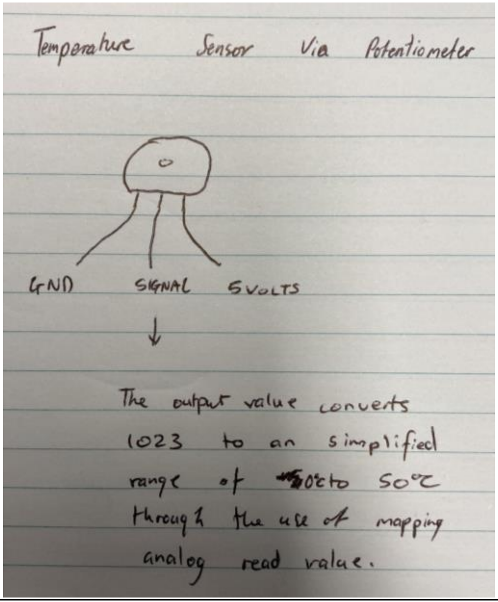
This project uses MQTT protocol to set up communication from Arduino to ThingsBoard via serial communication and Python script. When a publisher get data from Arduino, MQTT helps the data to be transferred into ThingsBoard. The subscriber fetches data from ThingsBoard using ThingsBoard Gateway API. MQTT protocol is set to perform to an interval of 60 seconds.

A brief rundown of our project

Equipment	***Purpose in the Project***	***Assigned Team Member***
Temperature Sensor	• Publisher • Sensor #1	Gurshan Dhaliwal
Humidity Sensor	• Publisher • Sensor #2	Bundy Huang
DC Motor Fan	• Subscriber • Actuator #1	Virul Vinwath
Buzzer	• Subscriber • Actuator #2	Le Bao Duy Nguyen

CONCEPTUAL DESIGN





TEAM STRUCTURE AND TASK BREAKDOWN

WEATHER STATION PROJECT

Name of Team Member	***Team Roles***
Gurshan Dhaliwal	• Report Formatting Documenting • Temperature Sensor (Through Potentiometer) • Fan Actuator • Video Upload to YouTube • Report Submission • ThingsBoard Configuration
Bundy Huang	• Buzzer/fan actuator • DHT11 sensor/ Potentiometer temperature sensor • Publish/Fetch data in Python and Arduino • Cloud Communication via ThingsBoard
Virul Vinwath	• Fan actuator • Transfer data from Python to Arduino
Le Bao Duy Nguyen	• Buzzer/fan actuator • DHT11 sensor/ Potentiometer temperature sensor • Cloud Communication via ThingsBoard • Publish/Fetch data in Python & Arduino • Gateway API and WebSocket API • Web interface • Report documenting

IMPLEMENTATION

SENSORS

Digital sensor DHT11: Measure humidity input. It can measure from 20-90% RH with an accuracy of +/-5%. This humidity acts as publisher data and is recorded in the database.

Potentiometer as Temperature sensor: Measure temperature input. This temperature acts as publisher data and is recorded in the database. Outputted data is within the pre-set range of 0 degrees to 50 degrees.

ACTUATORS

Buzzer: If temperature reaches below a certain threshold (i.e., 22 degrees Celsius), buzzer projects sound automatically. Buzzer can be manually turned on/off through updating the temperature value using command prompt.

Fan: If temperature reaches above a certain threshold (i.e., 22 degrees Celsius), fan spins automatically until it decreases to an adequate level. Fan can be manually turned on/off through updating the temperature value using command prompt.

VIRTUAL MACHINE (RASPBERRY PI)

Raspberry Pi acts as a host system that hosts the ThingsBoard database and the web interface. Python script is run to create the data as well as to fetch based on the serial communication received from Arduino.

MQTT PROTOCOL

When a publisher gets data from Arduino, MQTT helps the data to be transferred into ThingsBoard using a Python script that links to ThingsBoard IP address and device.

DATABASE

ThingsBoard is used to store data received from Arduino and the Raspberry Pi edge devices. 2 devices are used, 1 to record humidity and 1 to record temperature. Each device stores the data under attribute/telemetry records. Attribute data is fetched using ThingsBoard Gateway API in Python for Arduino to react, telemetry data is fetched using ThingsBoard WebSocket API to be used in the HTML/JavaScript web interface.

WEB INTERFACE

HTML/JavaScript interface that gets data from ThingsBoard via its WebSocket API to display the records of temperature and humidity recorded. If data crosses a threshold (i.e., above 22 degrees Celsius and/or humidity over 50%), the user is notified on the screen.

DATA VISUALISATION

ThingsBoard Dashboards are used to display temperature and humidity data recorded using column charts.

API

WebSocket API duplicates REST API functionality and provides the ability to subscribe to device telemetry data changes, then display in HTML/JavaScript interface.

ThingsBoard Gateway API allows fetching attributes data from ThingsBoard to Python via a script.

USER MANUAL

4 edge devices and 1 edge server (cloud) are configured. The edge devices communicate to the edge server via the edge server's ping.

Running temperature.ino and humidity.ino will start the recording of temperature and humidity data. Then temperature.py and humidity.py can be run to record those data into ThingsBoard's telemetry and attribute database located on the edge server by using MQTT protocol.

Temperature is recorded into Temperature Device with access token "TEMP" and humidity is recorded into Humidity Device with access token "HUMI".

By using Device MQTT API, users can fetch the data from the attribute database. Running buzzer.py and fan.py will execute the fetching action. Data will then be stripped into integers with a condition rule and passed using serial communication to the respective Arduino actuators. In buzzer.py, if the temperature is below 22 degrees Celsius, it will write "ON" to the Arduino, else if above it will write "OFF". fan.py will do the opposite, if the temperature is above 22 degrees Celsius, it will write "ON" to the Arduino, else if below it will write "OFF".

buzzer.ino and fan.ino will react based on the data received. If the temperature is below 22 degrees Celsius, the buzzer will receive "ON" and turn on whereas the fan will be off. If the temperature is above 22 degrees Celsius, the buzzer will receive "OFF" and turn off whereas the fan will be on.

HTML/JavaScript web pages will be divided into temperature.html and humidity.html. When run, the WebSocket API will get the data from the telemetry database of each device and display in a table form. Using JavaScript, a Device ID, JWT token and link to the ThingsBoard on the edge server are used to fetch the data, of which will be filtered to get only the needed integer of temperature/humidity to be displayed.

Bidirectional communication is done by not only getting the data using Arduino sensors, but users can also manually enter data into ThingsBoard's attribute using command prompt query, which then will be fetched, and the fan/buzzer can take action accordingly.

LIMITATIONS

LIMITATION OF THIS PROJECT IMPLEMENTATION

The limitations of this project were that group members did not have enough experience with interacting with things board interface and therefore many useful hours were spent trying to get our heads around the communication between raspberry pi and things board dashboard output. If in the future the team was to further develop this project, we would like to get physical boards to communicate over the network configuration (Wi-Fi) as it is easier to diagnose issues rather than trying to figure out how to work with the VirtualBox software's undescriptive and generic error codes.

ThingsBoard only stores the latest telemetry data of which the WebSocket API fetches to the HTML/JavaScript interface, hence our web interface can't display multiple data records recorded throughout the session. One way to overcome this is to store data once at a time into a database/Local Storage and fetch all the key-value pairs from there instead. However, this has not been done due to time constraint and availability.

Another limitation for this project is that the python scripts are not automatically/dynamically updated as temperature and humidity are adjusted; users must re-run the script in order to recapture the latest value from the sensors.

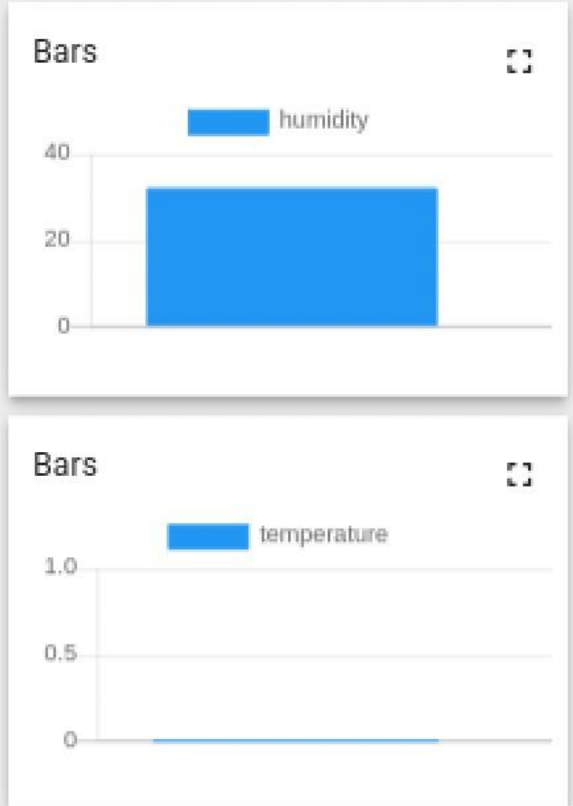
DEMONSTRATION VIDEO LINK

https://www.youtube.com/watch?v=8anmUGq7iiA

RESOURCES

1. https://pimylifeup.com/raspberry-pi-mosquitto-mqtt-server/
2. https://askubuntu.com/questions/1219498/could-not-open-port-dev-ttyacm0-error-after-every-restart
3. https://stackoverflow.com/questions/60802247/mqtt-subscriber-to-ThingsBoard-broker-in-python
4. https://create.arduino.cc/projecthub/pibots555/how-to-connect-dht11-sensor-with-arduino-uno-f4d239
5. https://ThingsBoard.io/docs/samples/raspberry/gpio/
6. https://ThingsBoard.io/docs/user-guide/install/rpi/
7. https://ThingsBoard.io/docs/user-guide/telemetry/
8. https://ThingsBoard.io/docs/reference/python-client-sdk/

APPENDIX

<table>
<tr><th>Image</th><th>Image Description</th></tr>
<tr><td>


</td><td>ThingsBoard Dashboard (Data Visualisation)</td></tr>
<tr><td>


</td><td>WebSocket API: uses device ID, JWT token and link to the ThingsBoard's device to fetch data from device's telemetry.</td></tr>
</table>

```
        webSocket.onmessage = function (event) {
            var received_msg = event.data;
            //alert("Message is received: " + received_msg);
            let tempData = [];
            tempData.push(JSON.parse(received_msg))
            console.log(tempData.map(x=>x.data))
            show(tempData.map(x=>x.data))
        };

        webSocket.onclose = function (event) {
            //alert("Connection is closed!");
        };
    }

    function show(data) {
  let tab =
    `<tr>
    <th>Temperature</th>
    </tr>`;

  // Loop to access all rows
  for (let r of data) {
    for(let t of r.temperature){
    tab += `<tr>
  <td>${t[1]} </td>

    </tr>`;
    if(t[1] >= 22) {
    warning = `<p style="color:red;">TOO HOT</p>`
    }
    else if (t[1] < 22) {
    warning = `<p>NORMAL</p>`
    }
  }}
  // Setting innerHTML as tab variable
  document.getElementById("temp").innerHTML = tab;
  document.getElementById("warning").innerHTML = warning;
}
  </script>

</head>
<body>

<div id="sse">
    <a href="javascript:WebSocketAPIExample()">Fetch Temperature</a>

    <table id="temp"></table>
    <div id="warning"></div>
</div>
```

Temperature HTML/JavaScript: fetches data by using WebSocket API.

<table>
<tr>
<td>

```
            webSocket.onmessage = function (event) {
                var received_msg = event.data;
                //alert("Message is received: " + received_msg);
                let tempData = [];
                tempData.push(JSON.parse(received_msg))
                console.log(tempData.map(x=>x.data))
                show(tempData.map(x=>x.data))
            };

            webSocket.onclose = function (event) {
                //alert("Connection is closed!");
            };
        }

        function show(data) {
    let tab =
        `<tr>
        <th>Humidity</th>
        </tr>`;

    // Loop to access all rows
    for (let r of data) {
        for(let t of r.humidity){
        tab += `<tr>
    <td>${t[1]} </td>

        </tr>`;
        if(t[1] >= 50) {
        warning = `<p style="color:red;">TOO HUMID</p>`
        }
        else if (t[1] < 50) {
        warning = `<p>NORMAL</p>`
        }
    }}
    // Setting innerHTML as tab variable
    document.getElementById("humidity").innerHTML = tab;
    document.getElementById("warning").innerHTML = warning;
}
    </script>

</head>
<body>

<div id="sse">
    <a href="javascript:WebSocketAPIExample()">Fetch Humidity</a>

    <table id="humidity"></table>
    <div id="warning"></div>
</div>
```

</td>
<td>Humidity HTML/JavaScript: fetches data by using WebSocket API.</td>
</tr>
<tr>
<td>mosquitto_pub -d -h "172.20.10.7" -t "v1/devices/me/attributes" -u "TEMP" -m "{"temperature": 20}"</td>
<td>Manual update of data to demonstrate bidirectional control via command prompt.</td>
</tr>
<tr>
<td>

```
import os
import time
import sys
import paho.mqtt.client as mqtt
import json
import serial


device = "/dev/ttyACM0"
arduino = serial.Serial(device,9600)
data = arduino.readline();
THINGSBOARD_HOST = '172.20.10.7'
ACCESS_TOKEN = 'HUMI'


# Data capture and upload interval in seconds. Less
interval will eventually hang the DHT22.
INTERVAL=2


sensor_data = {'humidity': data}


next_reading = time.time()


client = mqtt.Client()


# Set access token
client.username_pw_set(ACCESS_TOKEN)
# Connect to ThingsBoard using default MQTT port and 60
seconds keepalive interval
client.connect(THINGSBOARD_HOST, 1883, 60)
```

</td>
<td>Humidity Python script (publisher): to publish humidity data from Arduino to ThingsBoard device, both attribute and telemetry.</td>
</tr>
</table>

<pre>client.loop_start()
 	
try:
    while True:
                # Sending humidity and temperature data to
ThingsBoard
        client.publish('v1/devices/me/telemetry',
json.dumps(sensor_data), 1)
        client.publish('v1/devices/me/attributes',
json.dumps(sensor_data), 1)
        next_reading += INTERVAL
        sleep_time = next_reading-time.time()
        if sleep_time > 0:
            time.sleep(sleep_time)
            print(sensor_data)
except KeyboardInterrupt:
    pass
 	
client.loop_stop()
client.disconnect()</pre>	

<pre>import os
import time
import sys
import paho.mqtt.client as mqtt
import json
import serial
 	
device = "/dev/ttyACM0"
arduino = serial.Serial(device,9600)
data = arduino.readline();
THINGSBOARD_HOST = '172.20.10.7'
ACCESS_TOKEN = 'TEMP'
 	
# Data capture and upload interval in seconds. Less 	
interval will eventually hang the DHT22.
INTERVAL=2
 	
sensor_data = {'temperature': data}
 	
next_reading = time.time()
 	
client = mqtt.Client()
 	
# Set access token 	
client.username_pw_set(ACCESS_TOKEN)
 	
# Connect to ThingsBoard using default MQTT port and 60 	
seconds keepalive interval
client.connect(THINGSBOARD_HOST, 1883, 60)
 	
client.loop_start()
 	
try:
    while True:
                # Sending humidity and temperature data to
ThingsBoard
        client.publish('v1/devices/me/telemetry',
json.dumps(sensor_data), 1)
        client.publish('v1/devices/me/attributes',
json.dumps(sensor_data), 1)
        next_reading += INTERVAL
        sleep_time = next_reading-time.time()
        if sleep_time > 0:
            time.sleep(sleep_time)
            print(sensor_data)
except KeyboardInterrupt:
    pass
 	
client.loop_stop()
client.disconnect()</pre>
	Temperature Python script (publisher): to publish Temperature data from Arduino to ThingsBoard device, both attribute and telemetry.

humidity

```
#include <dht.h>

dht DHT;

#define DHT11_PIN 7

void setup(){
  Serial.begin(9600);
}

void loop(){
  int chk = DHT.read11(DHT11_PIN);
  Serial.println(DHT.humidity);
  delay(1000);
}
```

Humidity DHT11 sensor Arduino code (publisher): records humidity and pushes to Raspberry Pi.

```
// Declaring all used pins here
int tempPotentiometer = A0;
//int fan = 8;
int celsius = 0;

////DC Motor/Fan Operation Setup
//void fanOn(){
//  digitalWrite(fan, HIGH);
//}
//void fanOff(){
//  digitalWrite(fan, LOW);
//}

void setup()
{
//  pinMode(fan, OUTPUT);
  pinMode(tempPotentiometer, INPUT);
  Serial.begin(9600);
}

void loop(){
  // Temperature Boundaries and Calculation
  //map(variable, from min value, from max value, to min value, to max value)
  // celsius is capped at 50 degrees and min of 0 degrees
  celsius = map(((analogRead(A0))), 0, 1023, 0, 50);

  Serial.println(celsius);

//  if( celsius<= 22){
//    fanOff();
//  }
//  else if(celsius >= 23 && celsius <= 27){
//    fanOff();
//  }
//  else if(celsius > 27){
//    fanOn();
//  }
//  else{
//    Serial.println("00");
//  }
  delay(1000); // Give delay of 1 seconds before next value posted to Serial Monitor
}
```

Temperature Potentiometer Arduino code (publisher): records temperature and pushes to Raspberry Pi.

```
#define fan 2
String command;

void setup()
{
  Serial.begin(9600);
  pinMode(fan, OUTPUT);
}

void loop()
{
  if(Serial.available()){

   command = Serial.readStringUntil('\n');
   command.trim();
   if(command.equals("ON")){
    analogWrite(fan,255);
   }
   else if (command.equals("OFF")){
    analogWrite(fan,0);
   }
  }

   delay(1000);
}
```

Fan Arduino Code (subscriber): turn on when receive “ON” and off when receive “OFF”.

```
import logging
import time
import serial

from tb_device_mqtt import TBDeviceMqttClient
logging.basicConfig(level=logging.DEBUG)



def on_attributes_change(client, result, exception):
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 1)
    ser.flush()
    client.stop()

    while True:
        if exception is not None:
            print("Exception: " + str(exception))
        else:
            x = list(result.values())
            y = list(x[0].values())
            j = int(y[0])

        if __name__=='__main__':
            ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 1)
            ser.flush()
            print(j)

    #while True:
        if j >= 22:
            ser.write("ON\n".encode("utf-8"))
        if j < 22 :
            ser.write("OFF\n".encode("utf-8"))


def main():
    client = TBDeviceMqttClient("172.20.10.7", "TEMP")
    client.connect()
    client.request_attributes(["temperature", "temperature"], callback=on_attributes_change)
    while not client.stopped:
        time.sleep(1)


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

Fan Python Script (subscriber): to fetch the data from ThingsBoard’s attributes via Gateway API. If the temperature is over 22 degrees Celsius, print “ON” to the fan to make it turn on. Else if the temperature is below 22 degrees Celsius, print “OFF” to the fan to make it turn off.

```
void setup() {
  Serial.begin(9600);

  pinMode(BUZZER,OUTPUT);

}

void loop() {

  if(buzzer_state)
  {
analogWrite(BUZZER,constrain(buzzer_level,0,225));

  }

   if(Serial.available())
  {
    command=Serial.readStringUntil('\n');
    command.trim();
    if(command.equals("ON")){
    buzzer_state=true;

    }
    else if(command.equals("OFF")){

  digitalWrite(BUZZER,LOW);
   buzzer_state=false;
   //debugprintln("LED OFF");

   }
```

Buzzer Arduino Code (subscriber): turn on when receive "ON" and off when receive "OFF".

```python
import logging
import time
import serial

from tb_device_mqtt import TBDeviceMqttClient
logging.basicConfig(level=logging.DEBUG)



def on_attributes_change(client, result, exception):
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 1)
    ser.flush()
    client.stop()

    while True:
        if exception is not None:
            print("Exception: " + str(exception))
        else:
            x = list(result.values())
            y = list(x[0].values())
            j = int(y[0])

        if __name__=='__main__':
            ser = serial.Serial('/dev/ttyACM0', 9600, timeout = 1)
            ser.flush()
            print(j)

    #while True:
        if j <= 22:
            ser.write("ON\n".encode("utf-8"))
        if j > 22 :
            ser.write("OFF\n".encode("utf-8"))


def main():
    client = TBDeviceMqttClient("172.20.10.7", "TEMP")
    client.connect()
    client.request_attributes(["temperature", "temperature"], callback=on_attributes_change)
    while not client.stopped:
        time.sleep(1)


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

Buzzer Python Script (subscriber): to fetch the data from ThingsBoard's attributes via Gateway API. If the temperature is below 22 degrees Celsius, print "ON" to the buzzer to make it turn on. Else if the temperature is over 22 degrees Celsius, print "OFF" to the buzzer to make it turn off.