Vaasa University of Applied Sciences
IT2019-3C

BLE device and BLE gateway for IoT

Hoang Hieu - e1900314
Nguyen Hoang – e1900313
Hong Huynh – e1900318

Contents

1. Introduction

1.1 Purpose

To develop a small bluetooth device to monitoring environment sensing: Sound loudness, temperature, humidity, pressure, air quality... After that, Rasperry pi 4B was used as a BLE gateway to receive data from the device and using MQTT to publish and vizualize the data using Node-Red.

1.2 Scope

- Create a small and low power comsumption device.
- Able to collect and transmitting sensor data.
- Buzzer to alarm when detect smoke or bad air quality.
- Dashboard to manage data of the device and Rpi.
- Send alert email when detect loudness.
- Send alert email when device position been moved.

1.3 Definitions, Acronyms and Abbreviation

- **BLE** – Bluetooth Low Energy.
- **GATT** – Generic Attribute Profile.
- **IoT** – Internet of Things.
- **Gateway** – serves as the forwarding host (router) to other networks.
- **MQTT** – Message Queuing Telemetry Transport.
- **SSH** – Secure Socket Shell
- **UUID** – Universally unique identifier
- **RPi** – Raspberry Pi

1.4 Technologies to be used

Operating System	-	Window 10, Raspberry Pi OS.
Protocal	-	BLE GATT, MQTT.
Testing Application	-	nRF’s connect for mobile.
Development tool	-	Arduino IDE, Thonny Python IDE.
Dashboard development	-	Node-RED, Mosquitto MQTT.
PCB Designing tool	-	EasyEDA.

1.5 Overview

The bluetooth device made to monitor sound loudness and measuring environment sensings. It is useful to have a device to monitor for example in your rental apartment or airbnb. The device is small and low power consumption, works wired or wireless with rechargeable battery. Email alert and siren with security feature.

2. Overall setup description

2.1 Project overview

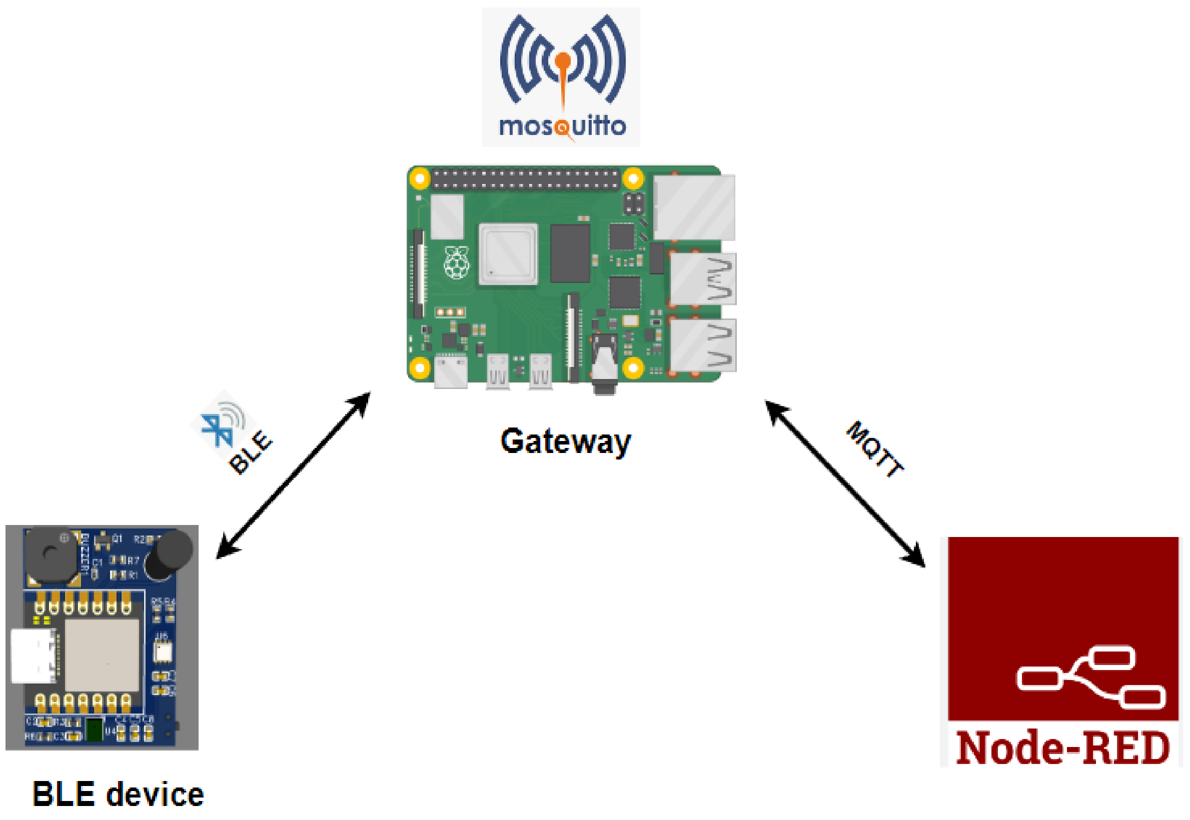


Figure 1

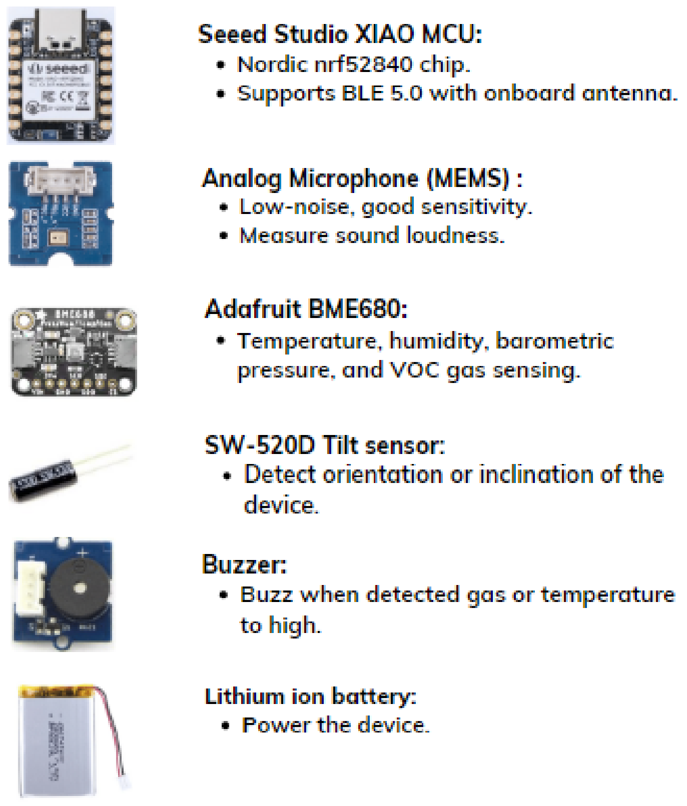
1. BLE device:
    - Responsible for collecting and transmitting sensors data over BLE.

2. Raspberry Pi as Gateway:
    - Responsible for retrieving/converting the data from BLE device and transmit data to the Cloud over Wi-fi or Ethernet.
    - Serve as MQTT broker using Mosquitto MQTT broker.

3. Node-RED:
    - Responsible for retrieving the data from Rpi through the MQTT protocal.
    - Vizualize the data in chart, graph also option to activate email alarms.

2.2 BLE device setup:

- List of components



Seeed Studio XIAO MCU:
- Nordic nrf52840 chip.
- Supports BLE 5.0 with onboard antenna.

Analog Microphone (MEMS) :
- Low-noise, good sensitivity.
- Measure sound loudness.

Adafruit BME680:
- Temperature, humidity, barometric pressure, and VOC gas sensing.

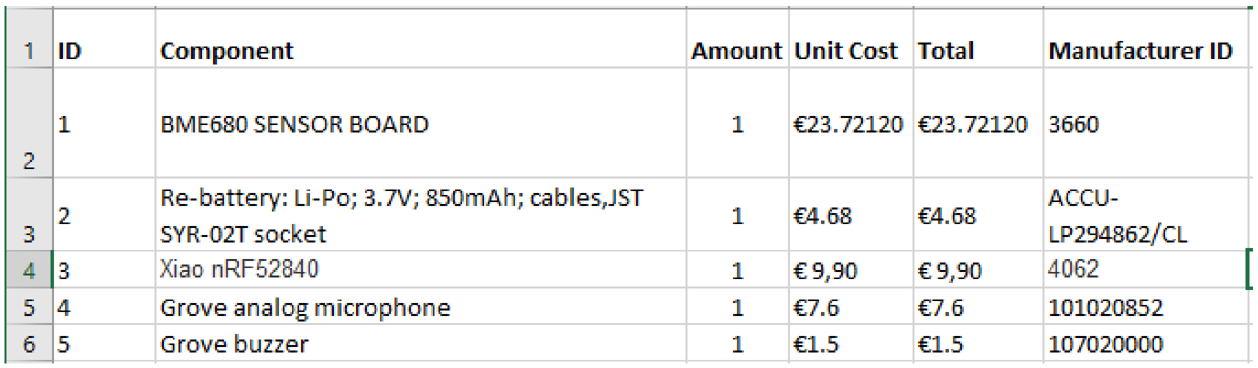
SW-520D Tilt sensor:
- Detect orientation or inclination of the device.

Buzzer:
- Buzz when detected gas or temperature to high.

Lithium ion battery:
- Power the device.

Figure 2

- Bill of material



1	ID	Component	Amount	Unit Cost	Total	Manufacturer ID
2	1	BME680 SENSOR BOARD	1	€23.72120	€23.72120	3660
3	2	Re-battery: Li-Po; 3.7V; 850mAh; cables,JST SYR-02T socket	1	€4.68	€4.68	ACCU-LP294862/CL
4	3	Xiao nRF52840	1	€ 9,90	€ 9,90	4062
5	4	Grove analog microphone	1	€7.6	€7.6	101020852
6	5	Grove buzzer	1	€1.5	€1.5	107020000

Figure 3

- Connections

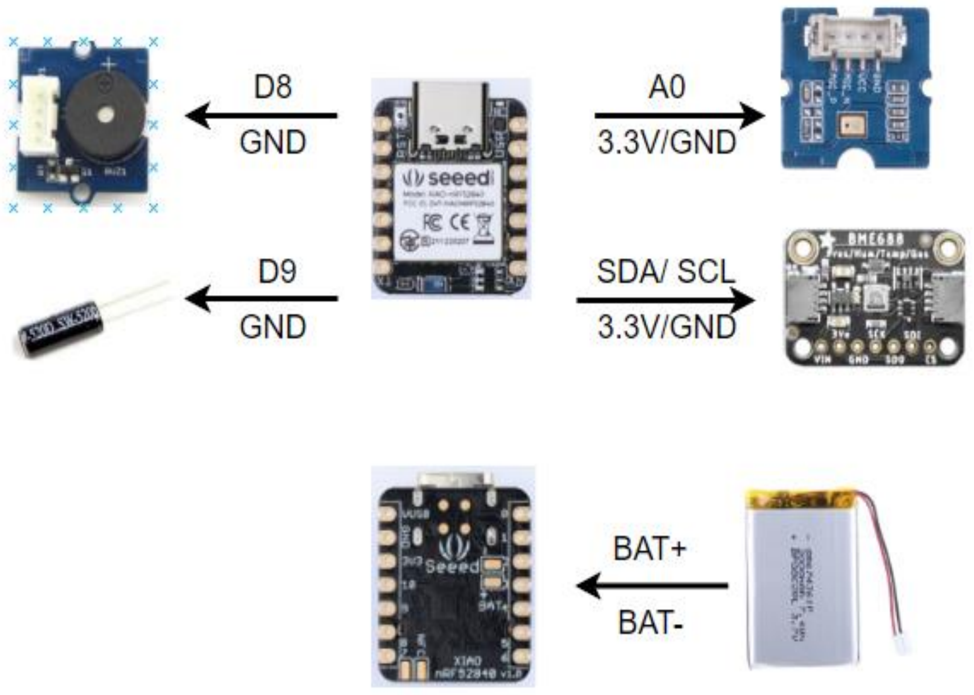


Figure 4

- Grove analog microphone to Xiao using analog Pin A0, VCC to 3.3V and GND to GND.

- BME680 to Xiao using I2C, SDA pin SDI, SCL to SCK, Vin to 3.3V and GND to GND.

- Grove buzzer to Xiao using digital pin D8 and the other pin to GND.

- SW 520D tilt sensor to digital pin D9 and the other pin to GND.

- Lithium battery soldered in bottom side of the Xiao to pin BAT +/ BAT -.

- Breadboard setup

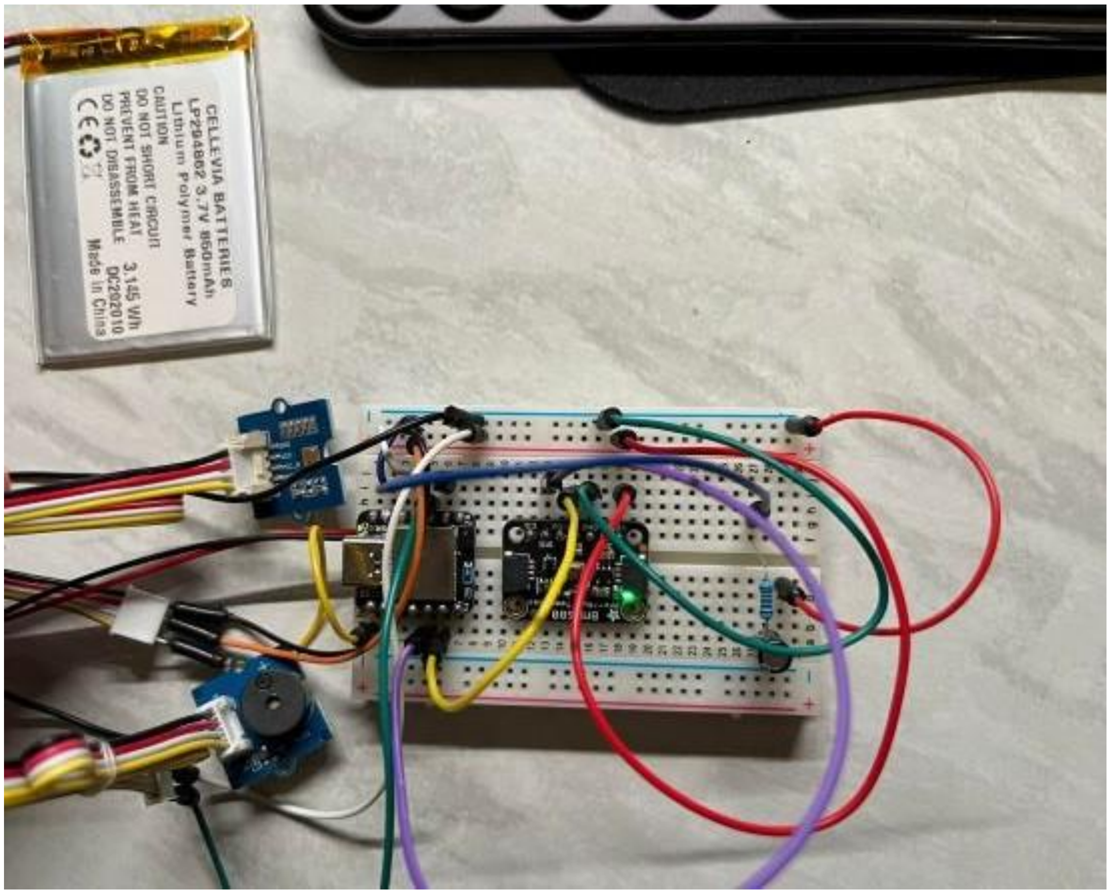


Figure 5

2.3 BLE Gateway setup

In this project, we will use a Raspberry pi model 4B to serve as a gateway. This Rpi model features a Quad-Core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, 4GB LPDDR4-2400 SDRAM, 2.4 GHz + 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet.

The BLE device will transmit IoT sensor telemetry, over BLE, to the Raspberry Pi. The Raspberry Pi, using Wi-Fi or Ethernet, is then able to securely transmit the sensor telemetry data to the Cloud. In Bluetooth terminology, the Bluetooth Peripheral device (aka GATT Server), which is the BLE device, will transmit data to the Bluetooth Central device (aka GATT Client), which is the Raspberry Pi.

2.4 Arduino sketch

For this post, the sketch, ‘main.ino’, contains all the code necessary to collect environmental sensor telemetry, including temperature, relative humidity, barometric pressure, and sound loudness.

```
#include <ArduinoBLE.h>
#include "Adafruit_BME680.h"
#define HICHG 22
#define BATR P0_14
#define buzzer 9
#define tiltpin 8
#define Gas_UUID "1be9c3a1-d3da-471b-8424-449a919af977"
#define Sound_UUID "782dcb14-2059-4e04-9baa-156188530999"
#define Tilt_UUID "ac5e30ce-ab84-42dd-ab0e-ed5c3903d0ee"

Adafruit_BME680 bme; // I2C

int previousTemperature = 0;
unsigned int previousHumidity = 0;
unsigned int previousPressure = 0;
unsigned int previousGas = 0;
unsigned int previousSound = 0;
const double vRef = 3.3; // Assumes 3.3V regulator output is ADC reference
voltage
const unsigned int numReadings = 1024; // 10-bit ADC readings 0-1023, so the
factor is 1024
long previousMillis = 0; // last time readings were checked, in ms
const int UPDATE_FREQUENCY = 5000;     // Update frequency in m

BLEService environmentService("181A"); // Standard Environmental Sensing
service

BLEService batteryService("180F"); // Battery servivce

BLEUnsignedCharCharacteristic batteryLevelChar("2A19", BLERead | BLENotify);

BLEIntCharacteristic tempCharacteristic("2A6E",BLERead | BLENotify);

BLEUnsignedIntCharacteristic humidCharacteristic("2A6F",BLERead | BLENotify);

BLEUnsignedIntCharacteristic pressureCharacteristic("2A6D",BLERead |
BLENotify);

BLEUnsignedIntCharacteristic gasCharacteristic(Gas_UUID,BLERead | BLENotify);

BLEUnsignedIntCharacteristic soundCharacteristic(Sound_UUID,BLERead |
BLENotify);

BLEIntCharacteristic tiltCharacteristic(Tilt_UUID,BLERead | BLENotify);

void  init_sensor (Adafruit_BME680& bme) {
  bme.setTemperatureOversampling (BME680_OS_8X);
  bme.setHumidityOversampling (BME680_OS_2X);
  bme.setPressureOversampling (BME680_OS_4X);
  bme.setIIRFilterSize (BME680_FILTER_SIZE_3);
  bme.setGasHeater ( 320 , 150 ); // 320*C for 150 milliseconds
}

void setup() {
  Serial.begin(115200);

  if (!bme.begin()) {
    while (1);
  }
  init_sensor (bme);

  pinMode(LED_BUILTIN, OUTPUT);
  pinMode (HICHG, OUTPUT);  digitalWrite(HICHG, LOW); // HIGH Charging mode

  if (!BLE.begin()) {
    while (1);
  }
  BLE.setLocalName("nrf52840");    // Set name for connection
  BLE.setAdvertisedService(environmentService); // Advertise environment
service
//  BLE.setAdvertisedService(batteryService);

  batteryService.addCharacteristic(batteryLevelChar);
  environmentService.addCharacteristic(tempCharacteristic);     // Add
temperature characteristic
  environmentService.addCharacteristic(humidCharacteristic);    // Add
humidity characteristic
  environmentService.addCharacteristic(pressureCharacteristic); // Add
pressure characteristic
  environmentService.addCharacteristic(gasCharacteristic);      // Add gas
characteristic
  environmentService.addCharacteristic(soundCharacteristic);    // Add sound
characteristic
  environmentService.addCharacteristic(tiltCharacteristic);    // Add tilt
characteristic

  BLE.addService(batteryService);
  BLE.addService(environmentService); // Add environment service

  tempCharacteristic.setValue(0);     // Set initial temperature value
  humidCharacteristic.setValue(0);    // Set initial humidity value
  pressureCharacteristic.setValue(0); // Set initial pressure value
  gasCharacteristic.setValue(0);      // Set initial gas value
  soundCharacteristic.setValue(0);    // Set initial sound value
  tiltCharacteristic.setValue(0);     // Set initial tilt value
  batteryLevelChar.writeValue(0);

  BLE.advertise(); // Start advertising
  Serial.println("Bluetooth® device active, waiting for connections...");
}


void loop() {
   BLEDevice central = BLE.central();
   if (central) {
        Serial.print("Connected to central MAC: ");
        Serial.println(central.address()); // Central's BT address:

        digitalWrite(LED_BUILTIN, HIGH); // Turn on the LED to indicate the
connection

        while (central.connected()) {
            long currentMillis = millis();
            // After UPDATE_FREQUENCY ms have passed, check temperature &
humidity
            if (currentMillis - previousMillis >= UPDATE_FREQUENCY) {
                previousMillis = currentMillis;
                updateReadings();
                updateBatteryLevel();
                alarm();
            }
        }

        digitalWrite(LED_BUILTIN, LOW); // When the central disconnects, turn
off the LED
        Serial.print("Disconnected from central MAC: ");
        Serial.println(central.address());
    }
}

void updateReadings() {
    int temperature = bme.readTemperature() * 100;
    unsigned int humidity = bme.readHumidity() * 100;
    unsigned int pressure = bme.readPressure();
    unsigned int gas = bme.readGas();
    unsigned int sound = (analogRead(A0) + 83.2073) /3.503;
    int tilt = digitalRead(tiltpin);

    if (temperature != previousTemperature) { // If reading has changed
        Serial.print("Temperature: ");
        Serial.println(temperature);
        tempCharacteristic.writeValue(temperature); // Update characteristic
        previousTemperature = temperature;          // Save value
    }

    if (humidity != previousHumidity) { // If reading has changed
        Serial.print("Humidity: ");
        Serial.println(humidity);
        humidCharacteristic.writeValue(humidity);
        previousHumidity = humidity;
    }

    if (pressure != previousPressure) { // If reading has changed
        Serial.print("Pressure: ");
        Serial.println(pressure);
        pressureCharacteristic.writeValue(pressure);
        previousPressure = pressure;
    }
    if (gas != previousGas) { // If reading has changed
        Serial.print("Gas: ");
        Serial.println(gas);
        gasCharacteristic.writeValue(gas);
        previousGas = gas;
    }

    if (sound != previousSound) { // If reading has changed
        Serial.print("Sound(dB): ");
        Serial.println(sound);
        soundCharacteristic.writeValue(sound);
        previousSound = sound;
    }

     tiltCharacteristic.writeValue(tilt);
     delay(50);
}

void updateBatteryLevel() {
  unsigned int adcCount = analogRead(PIN_VBAT);
  double adcVoltage = (adcCount * vRef) / numReadings;
  double vBat =(adcVoltage*1550.0/510.0) *100; // Voltage divider from Vbat
to ADC
  unsigned int BatteryLevel;

  if( 4 <= vBat){
    BatteryLevel= 100;
    batteryLevelChar.writeValue(BatteryLevel);
    Serial.print("Battery Level % is now: "); // print it
    Serial.println(BatteryLevel);
  }
  else if( 3.93<= vBat <= 4){
    BatteryLevel= 90;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.89<= vBat <= 3.93){
    BatteryLevel= 80;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.85<= vBat <= 3.89){
    BatteryLevel= 70;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.82<= vBat <= 3.85){
    BatteryLevel= 60;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.79<= vBat <= 3.82){
    BatteryLevel= 50;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.75<= vBat <= 3.79){
    BatteryLevel= 40;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.72<= vBat <= 3.75){
    BatteryLevel= 30;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.7<= vBat <= 3.72){
    BatteryLevel= 20;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if( 3.65<= vBat <= 3.7){
    BatteryLevel= 10;
    batteryLevelChar.writeValue(BatteryLevel);
  }
  else if(vBat <= 3.65){
    BatteryLevel= 0;
    batteryLevelChar.writeValue(BatteryLevel);
  }
}

void alarm()  {
  int temp = bme.readTemperature();
  unsigned int humid = bme.readHumidity();
  unsigned int Gas = bme.readGas();
  if(temp >60 && humid <10 || Gas < 5000){
    tone(buzzer, 400, 500); //the buzzer emit sound at 400 MHz for 500 millis
    delay(500); //wait 500 millis
    tone(buzzer, 650, 500); //the buzzer emit sound at 650 MHz for 500 millis
    delay(500); //wait 500 millis
  }
}
```

The sensor telemetry will be advertised by the device, over BLE, as a GATT Environmental Sensing Service (GATT Assigned Number 0x181A) with multiple GATT Characteristics. Each Characteristic represents a sensor reading and contains the most current sensor value(s), for example, Temperature (0x2A6E) or Humidity (0x2A6F).

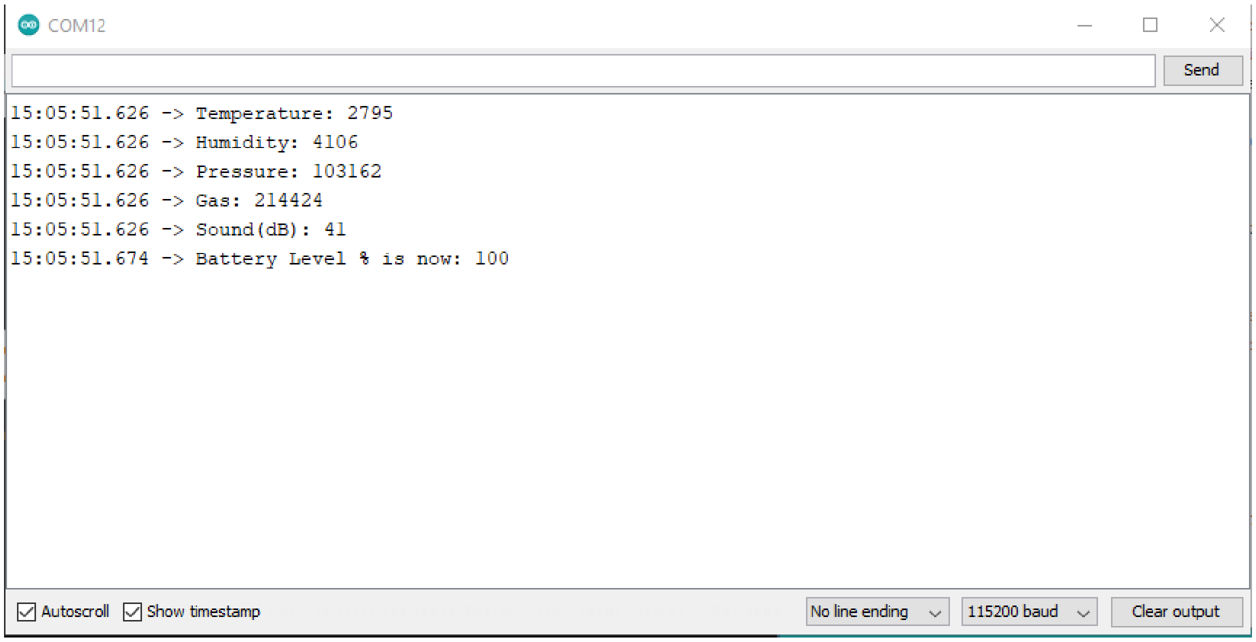
Each GATT Characteristic defines how the data should be represented. To represent the data accurately, the sensor readings need to be modified. For example, using Adafruit-BME680 library, the temperature is captured with two decimal points of precision (e.g., 22.21 °C). However, the Temperature GATT Characteristic (0x2A6E) requires a signed 16-bit value (-32,768–32,767). To maintain precision, the captured value (e.g., 22.21 °C) is multiplied by 100 to convert it to an integer (e.g., 2221). The Raspberry Pi will then handle converting the value back to the original value with the correct precision.

The GATT specification has no current predefined Characteristic representing the sound, gas so we have created a custom Characteristic for these values and assigned it a universally unique identifier (UUID).

```
BLEService environmentService("181A"); // Standard Environmental Sensing service

BLEService batteryService("180F"); // Battery servivce

BLEUnsignedCharCharacteristic batteryLevelChar("2A19", BLERead | BLENotify);

BLEIntCharacteristic tempCharacteristic("2A6E",BLERead | BLENotify);

BLEUnsignedIntCharacteristic humidCharacteristic("2A6F",BLERead | BLENotify);

BLEUnsignedIntCharacteristic pressureCharacteristic("2A6D",BLERead | BLENotify);

BLEUnsignedIntCharacteristic gasCharacteristic(Gas_UUID,BLERead | BLENotify);

BLEUnsignedIntCharacteristic soundCharacteristic(Sound_UUID,BLERead | BLENotify);
```

Figure 6

- Data retrieved in Arduino Serial Monitor

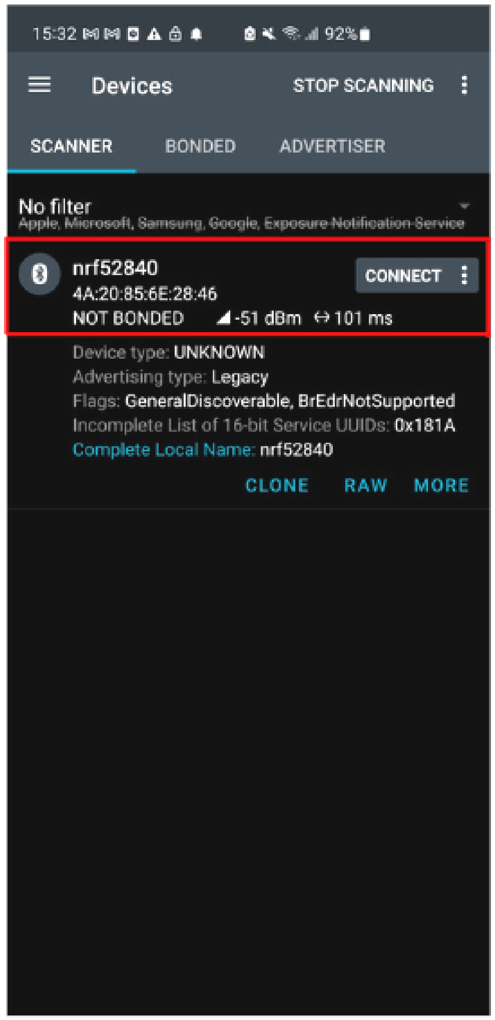


```
15:05:51.626 -> Temperature: 2795
15:05:51.626 -> Humidity: 4106
15:05:51.626 -> Pressure: 103162
15:05:51.626 -> Gas: 214424
15:05:51.626 -> Sound(dB): 41
15:05:51.674 -> Battery Level % is now: 100
```

Figure 7

2.5 Previewing BLE Device Services

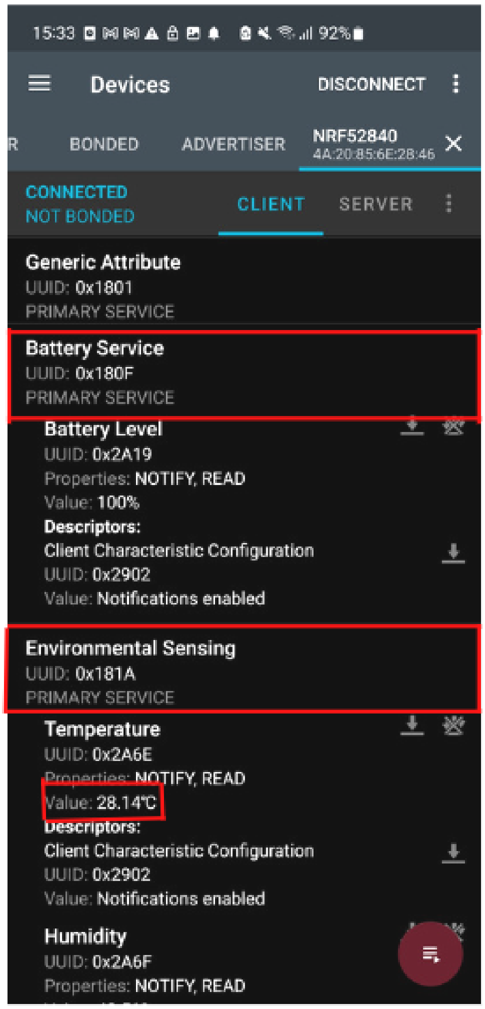
Before looking at the code running on the Raspberry Pi, we can use any number of mobile applications to preview and debug the Environmental Sensing, Battery service running on the deviceand being advertised over BLE. A commonly recommended application is Nordic Semiconductor's nRF Connect for Mobile.



No filter
Apple, Microsoft, Samsung, Google, Exposure Notification Service

nrf52840
4A:20:85:6E:28:46
NOT BONDED -51 dBm ↔ 101 ms

Device type: UNKNOWN
Advertising type: Legacy
Flags: GeneralDiscoverable, BrEdrNotSupported
Incomplete List of 16-bit Service UUIDs: 0x181A
Complete Local Name: nrf52840



CONNECTED
NOT BONDED

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Battery Service
UUID: 0x180F
PRIMARY SERVICE

Battery Level
UUID: 0x2A19
Properties: NOTIFY, READ
Value: 100%
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

Environmental Sensing
UUID: 0x181A
PRIMARY SERVICE

Temperature
UUID: 0x2A6E
Properties: NOTIFY, READ
Value: 28.14℃
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

Humidity
UUID: 0x2A6F
Properties: NOTIFY, READ

These results indicate everything is working as expected.

2.6 BLE gateway python script

To act as the BLE Client (aka central device), the Raspberry Pi runs a Python script. The script uses the bluepy Python module for interfacing with BLE devices through Bluez, on Rpi OS.

```
import sys
import time
import paho.mqtt.client as mqtt
from argparse import ArgumentParser
from bluepy import btle
#4A:20:85:6E:28:46

mqttBroker ="192.168.0.103"
client = mqtt.Client("Gateway")
client.connect(mqttBroker, 1883)

def main():
    # get args
    args = get_args()

    print("Connecting...")
    ble_gateway = btle.Peripheral(args.mac_address)

    print("Discovering Services...")
    _ = ble_gateway.services
    environmental_sensing_service = ble_gateway.getServiceByUUID("181A")
    battery_service = ble_gateway.getServiceByUUID("180F")

    print("Discovering Characteristics...")
    _ = environmental_sensing_service.getCharacteristics()
    _ = battery_service.getCharacteristics()

    while True:
        print("\n")
        read_temperature(environmental_sensing_service)
        read_humidity(environmental_sensing_service)
        read_pressure(environmental_sensing_service)
        read_gas(environmental_sensing_service)
        read_sound(environmental_sensing_service)
        read_tilt(environmental_sensing_service)
        read_BAT(battery_service)
        time.sleep(2)


def byte_array_to_int(value):
    value = bytearray(value)
    value = int.from_bytes(value, byteorder="little")
    return value


def decimal_exponent_two(value):
    return value / 100
```

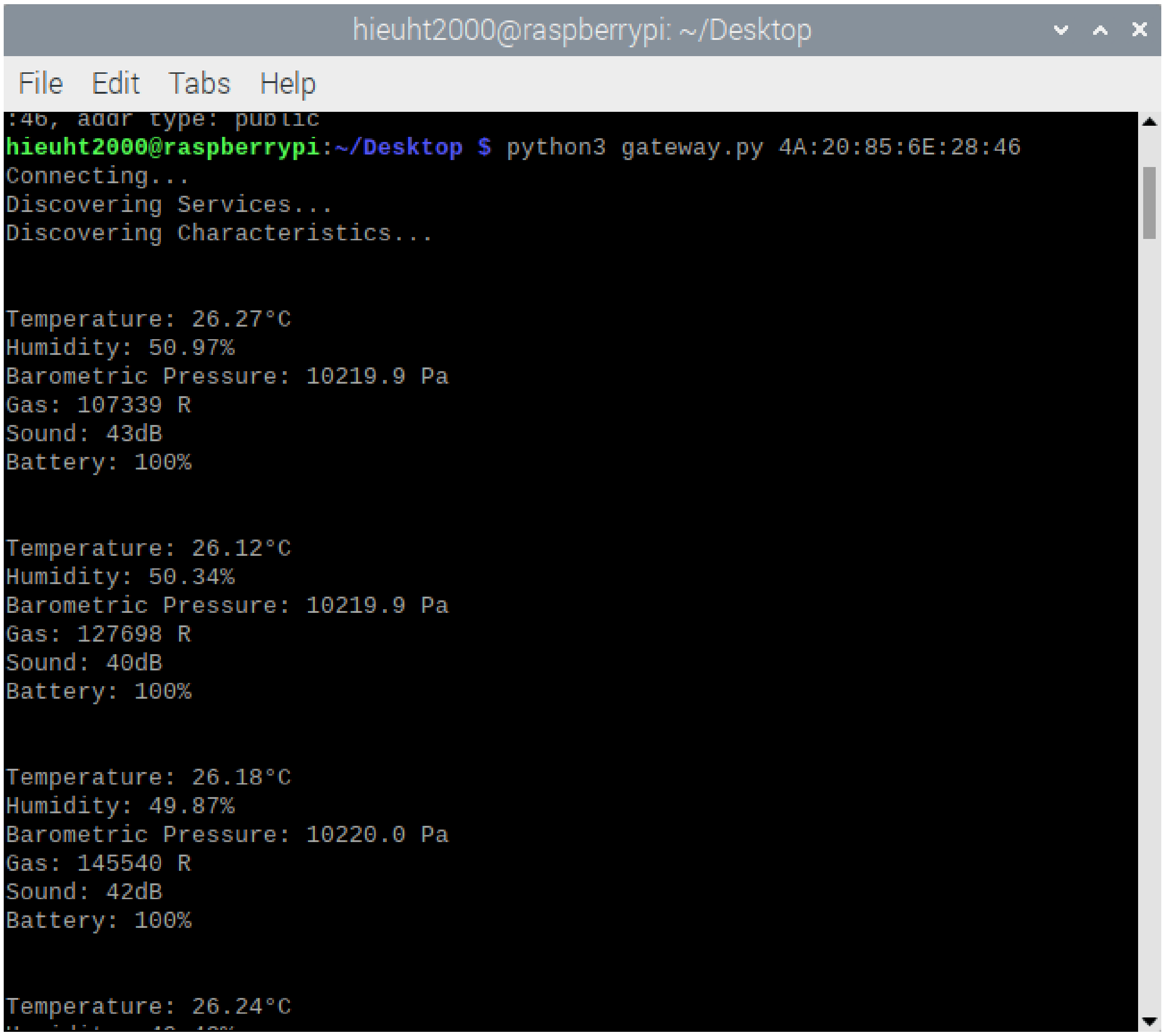
```
def decimal_exponent_one(value):
    return value / 10


def read_pressure(service):
    pressure_char = service.getCharacteristics("2A6D")[0]
    pressure = pressure_char.read()
    pressure = byte_array_to_int(pressure)
    pressure = decimal_exponent_one(pressure)
    client.publish("gateway/pressure", pressure)
    print(f"Barometric Pressure: {round(pressure, 2)} Pa")


def read_humidity(service):
    humidity_char = service.getCharacteristics("2A6F")[0]
    humidity = humidity_char.read()
    humidity = byte_array_to_int(humidity)
    humidity = decimal_exponent_two(humidity)
    client.publish("gateway/humidity", humidity)
    print(f"Humidity: {round(humidity, 2)}%")


def read_temperature(service):
    temperature_char = service.getCharacteristics("2A6E")[0]
    temperature = temperature_char.read()
    temperature = byte_array_to_int(temperature)
    temperature = decimal_exponent_two(temperature)
    client.publish("gateway/temperature", temperature)
    print(f"Temperature: {round(temperature, 2)}°C")

def read_gas(service):
    gas_char = service.getCharacteristics("1be9c3a1-d3da-471b-8424-
449a919af977")[0]
    gas = gas_char.read()
    gas = byte_array_to_int(gas)
    client.publish("gateway/gas", gas)
    print(f"Gas: {round(gas)} R")

def read_sound(service):
    sound_char = service.getCharacteristics("782dcb14-2059-4e04-9baa-
156188530999")[0]
    sound = sound_char.read()
    sound = byte_array_to_int(sound)
    client.publish("gateway/sound", sound)
    print(f"Sound: {round(sound)}dB")

def read_tilt(service):
    previousTilt = 0
    tilt_char = service.getCharacteristics("ac5e30ce-ab84-42dd-ab0e-
ed5c3903d0ee")[0]
    tilt = tilt_char.read()
    tilt = byte_array_to_int(tilt)
    client.publish("gateway/tilt", tilt)
    if tilt != previousTilt:
        print("Position changed")
```

```
def read_BAT(service):
    bat_char = service.getCharacteristics("2A19")[0]
    bat = bat_char.read()
    bat = byte_array_to_int(bat)
    client.publish("gateway/battery", bat)
    print(f"Battery: {round(bat)}%")

def get_args():
    arg_parser = ArgumentParser(description="Gateway")
    arg_parser.add_argument('mac_address', help="MAC address of device to
connect")
    args = arg_parser.parse_args()
    return args


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

To run the Python script, execute the following command:

```
Python3 gateway.py 4A:20:85:6E:28:46
```

The script takes the raw, incoming hexadecimal text from the Arduino and coerces it to the correct values. For example, a temperature reading must be transformed from bytes, b'\xb8\x08\x00\x00', to a byte array, bytearray(b'\xb8\x08\x00\x00'), then to an integer, 2232, then to a decimal, 22.32. Sensor readings are retrieved from the BLE device every two seconds.



```
:46, addr type: public
hieuht2000@raspberrypi:~/Desktop $ python3 gateway.py 4A:20:85:6E:28:46
Connecting...
Discovering Services...
Discovering Characteristics...


Temperature: 26.27°C
Humidity: 50.97%
Barometric Pressure: 10219.9 Pa
Gas: 107339 R
Sound: 43dB
Battery: 100%


Temperature: 26.12°C
Humidity: 50.34%
Barometric Pressure: 10219.9 Pa
Gas: 127698 R
Sound: 40dB
Battery: 100%


Temperature: 26.18°C
Humidity: 49.87%
Barometric Pressure: 10220.0 Pa
Gas: 145540 R
Sound: 42dB
Battery: 100%


Temperature: 26.24°C
```

Figure 8

2.7 MQTT Protocal

Install Mosquitto MQTT broker on RPI

- You should have the Raspberry Pi OS installed in your Raspberry Pi – Install Raspberry Pi OS, Set Up Wi-Fi, Enable and Connect with SSH;
- You also need the following hardware:
  - Raspberry Pi board – read Best Raspberry Pi Starter Kits.
  - MicroSD Card – 16GB Class10.
  - Raspberry Pi Power Supply (5V 2.5A).

- To install the Mosquitto Broker enter these next commands:

```
sudo apt install -y mosquitto mosquitto-clients
```

- To make Mosquitto auto start when the Raspberry Pi boots, you need to run the following command (this means that the Mosquitto broker will automatically start when the Raspberry Pi starts):

```
sudo systemctl enable mosquitto.service
```

- Now, test the installation by running the following command:

```
mosquitto -v
```

This returns the Mosquitto version that is currently running in your Raspberry Pi. It will be 2.0.11 or above.

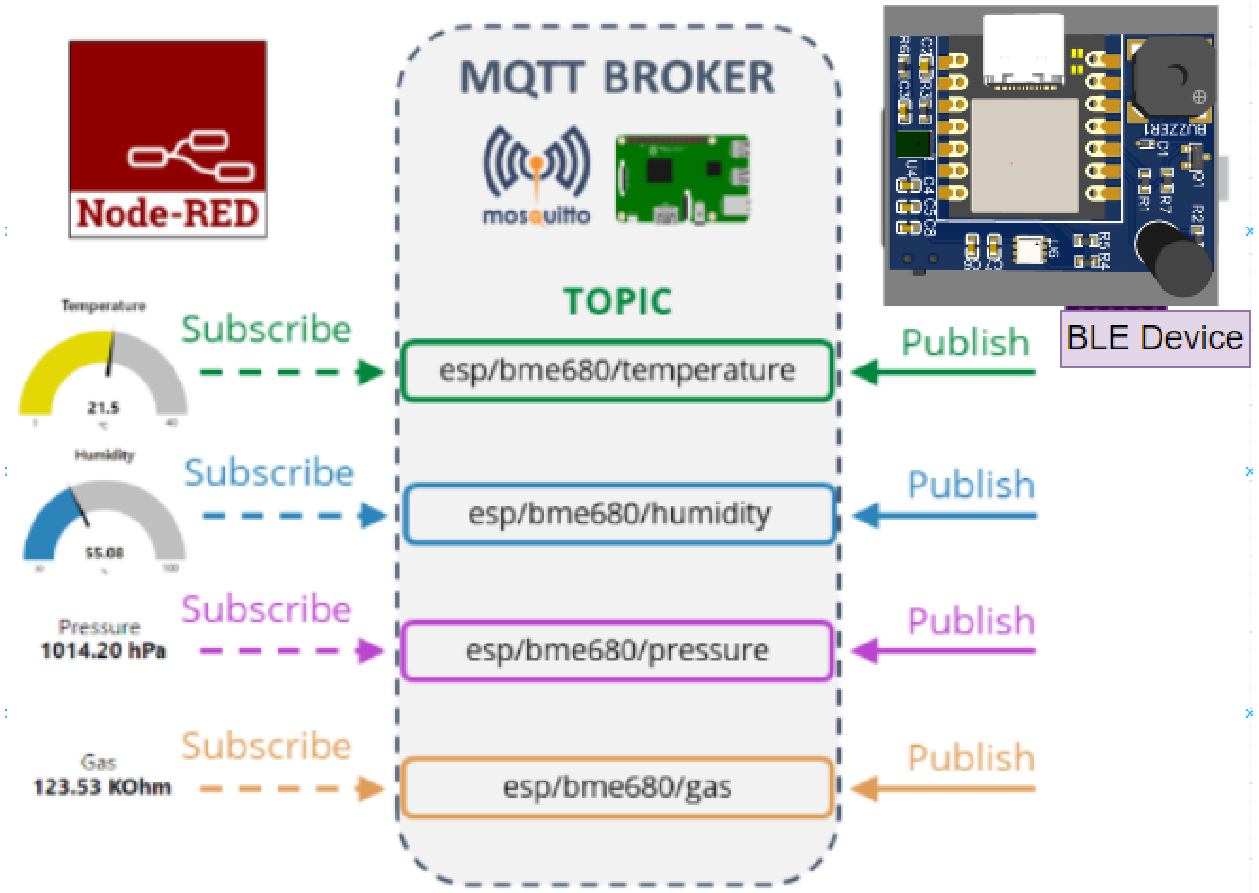


Figure 9

The Python script published sensor data to Mosquitto MQTT Broker and Node-RED retrieved and vizualize the data.

3. Node-RED Dashboard features

3.1 Dashboard Flow-chart

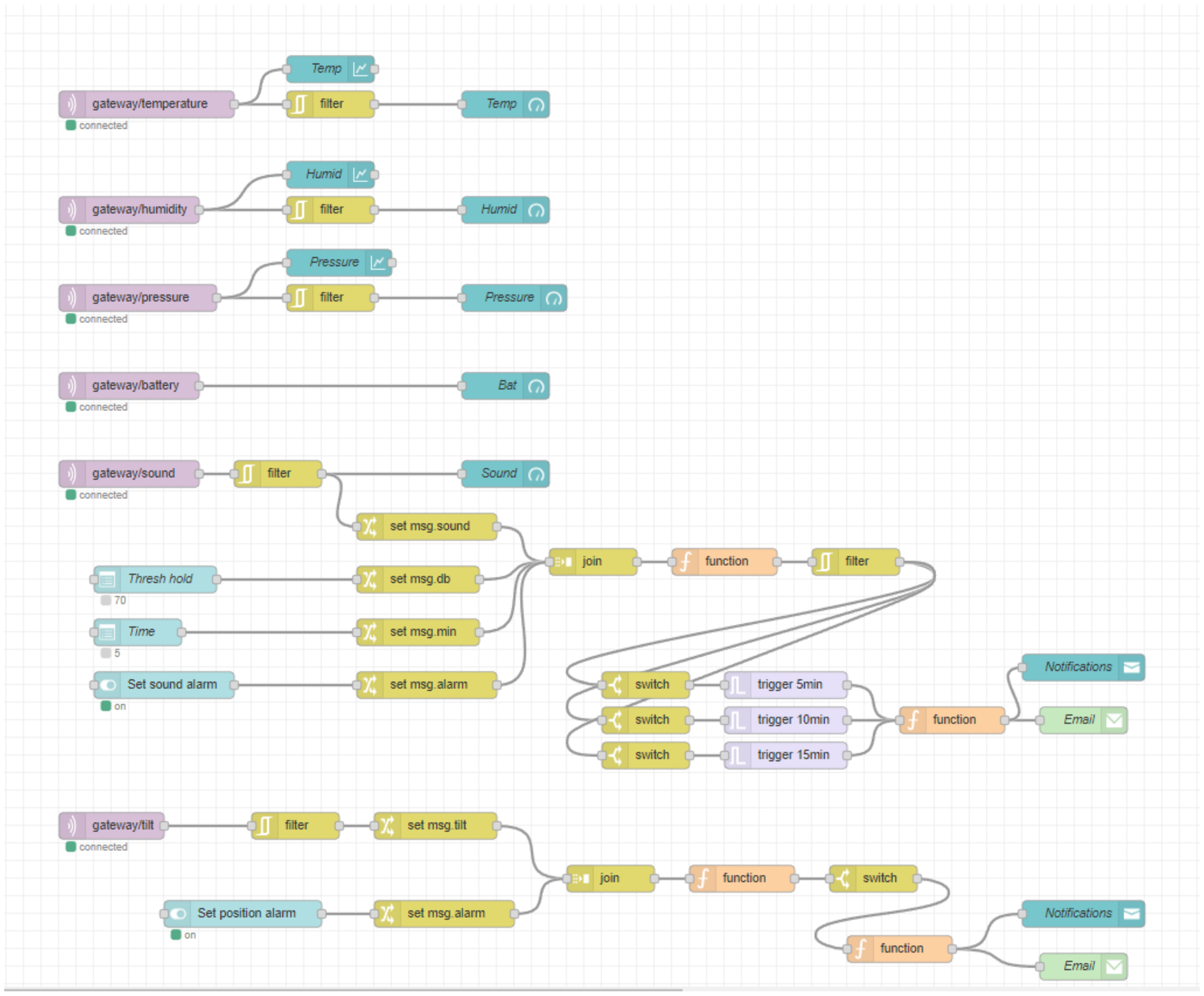


Figure 10

- We used MQTT nodes to retrived the data and used dashboard node to vizualize data.

3.2 Dashboard features

- **Overall Dashboard**

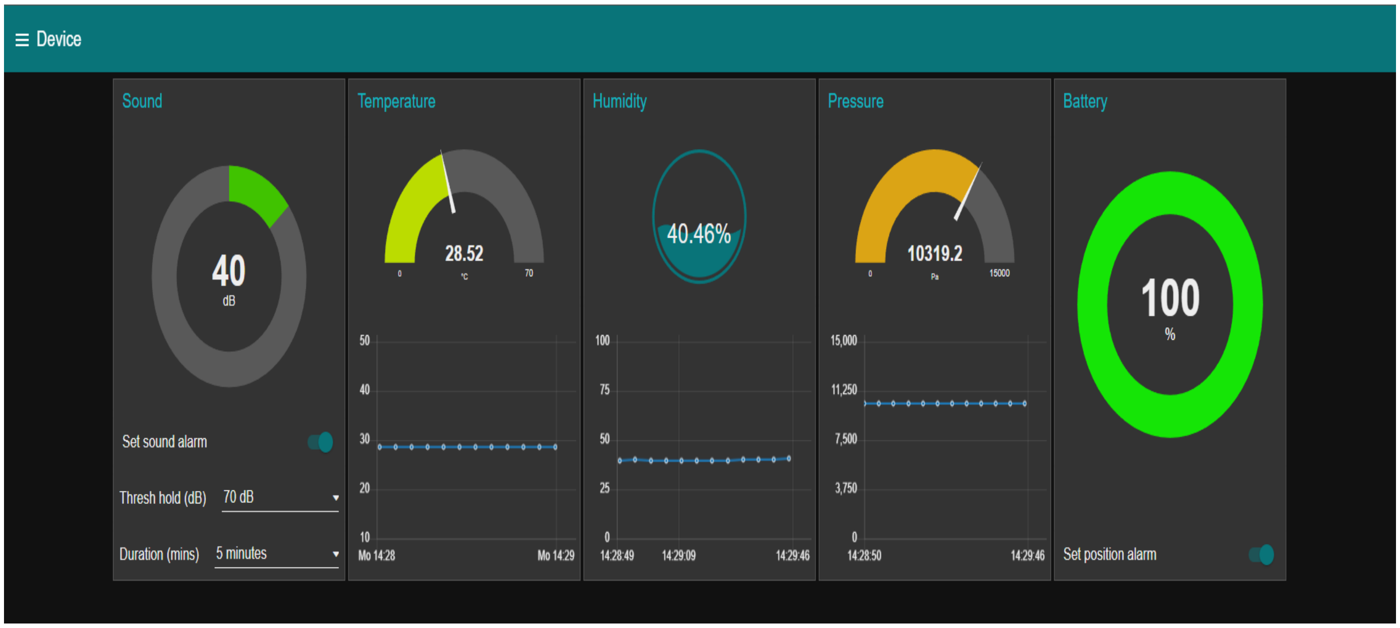
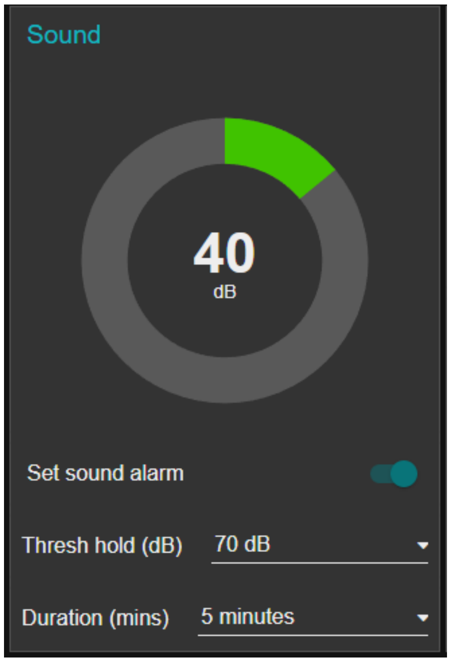


Figure 11

- **Email alarm features**



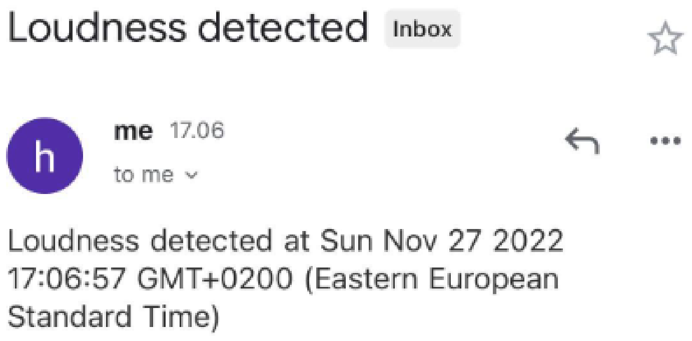
Thresh hold (dB) 70 dB

Duration (mins) 5 minutes

Figure 12

You can turn on /off the alarm, set the value for thresh hold and duration.

For example, if sound is over 70dB for 5 minutes it will send you an email.


Loudness detected Inbox

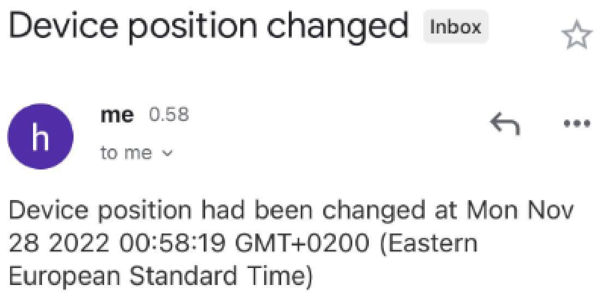
me 17.06
to me

Loudness detected at Sun Nov 27 2022 17:06:57 GMT+0200 (Eastern European Standard Time)

Figure 13

Position alarm: will send you an email if device been moved.

Figure 14


Device position changed Inbox

me 0.58
to me

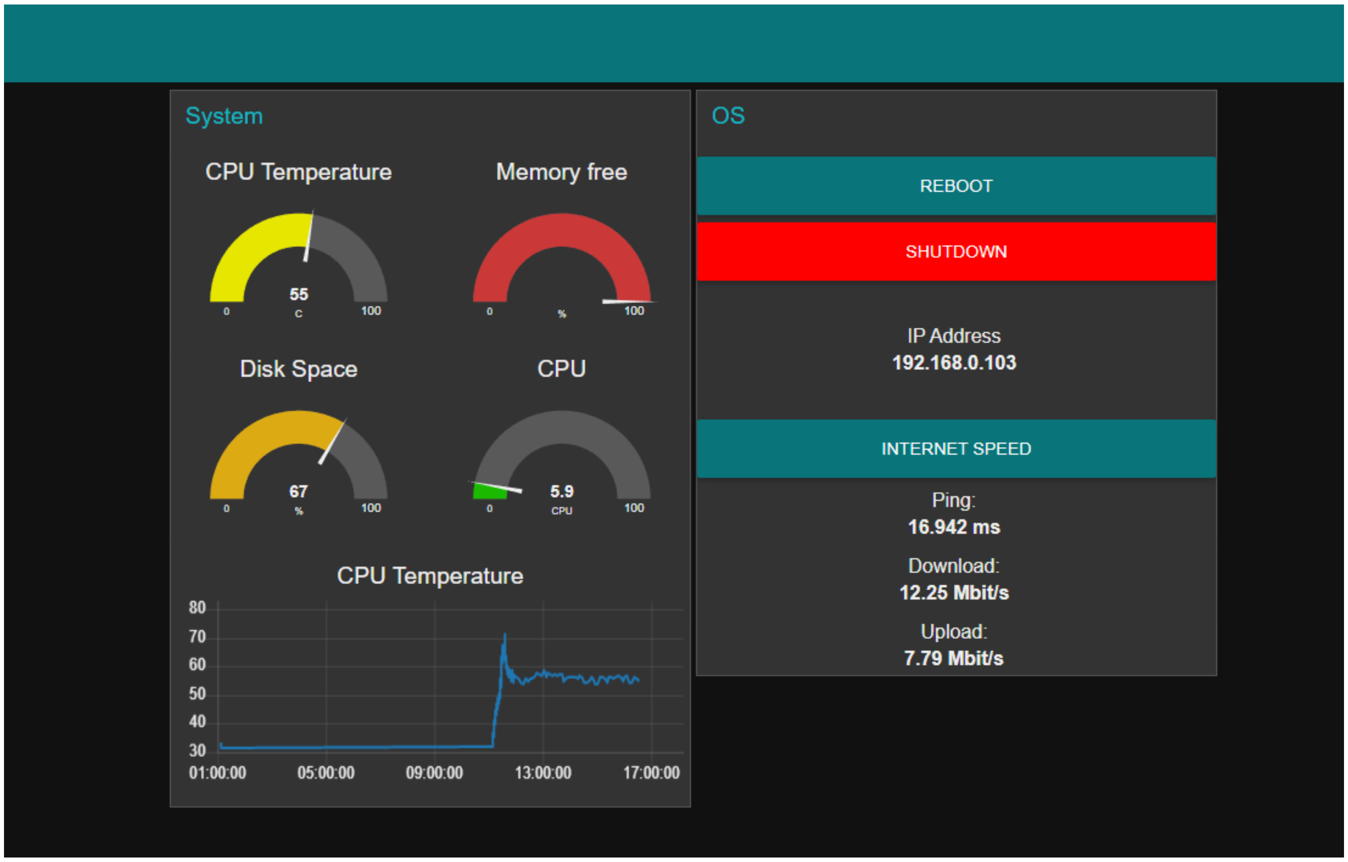
Device position had been changed at Mon Nov 28 2022 00:58:19 GMT+0200 (Eastern European Standard Time)

Figure 15

- **Dashboard to monitor Rpi**

You can manage the data of your Rpi: CPU temperature, RAM, Disk memory.

Button to Reboot, Shutdown, get Internet Speed.



IP Address
192.168.0.103

Ping:
16.942 ms

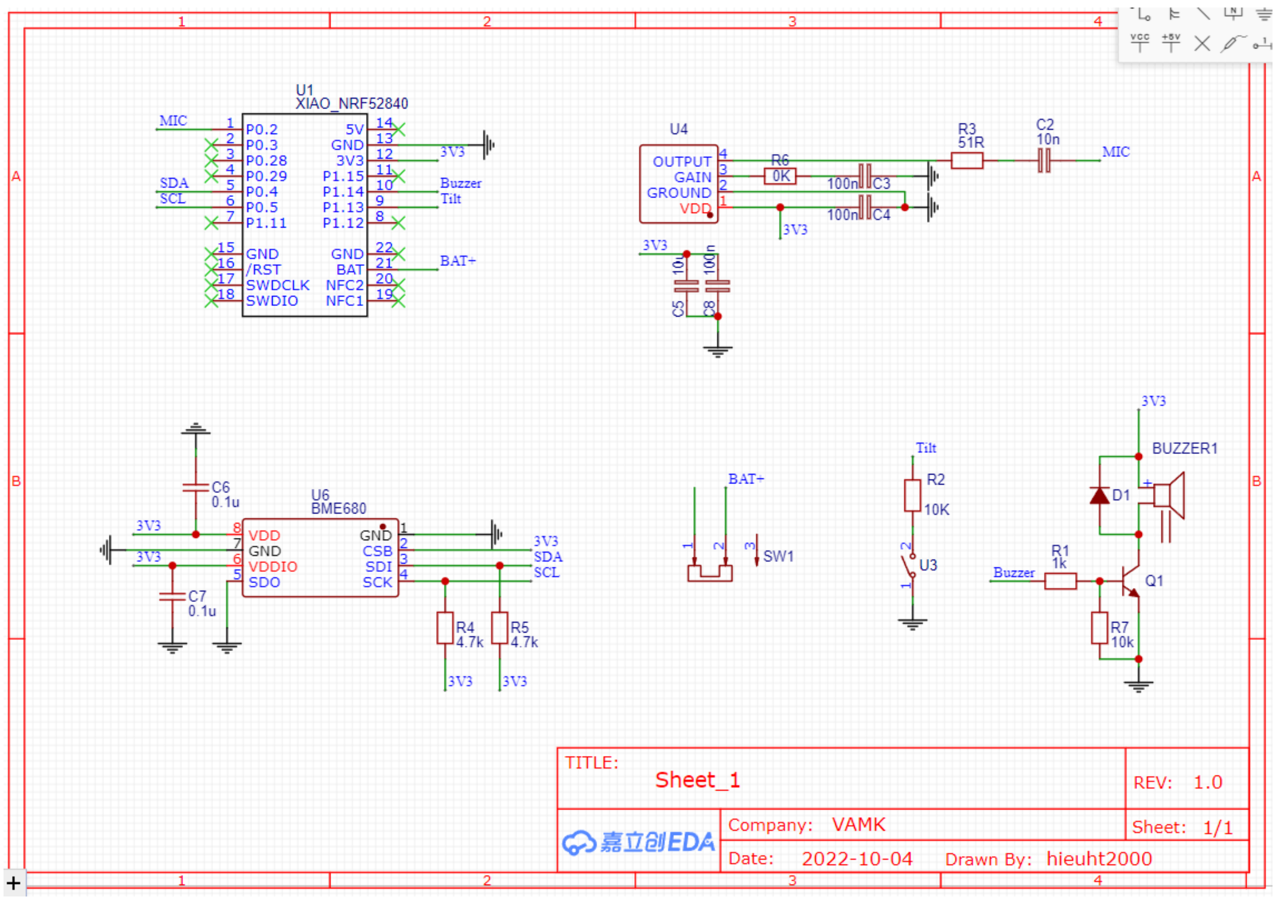
Download:
12.25 Mbit/s

Upload:
7.79 Mbit/s

Figure 16

4. PCB design

- Schematic



Footprint:

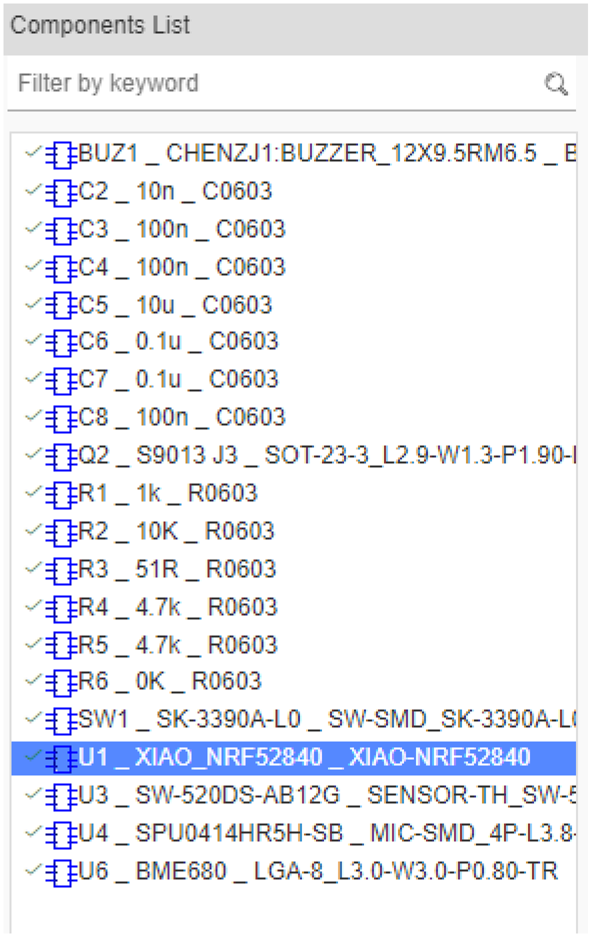
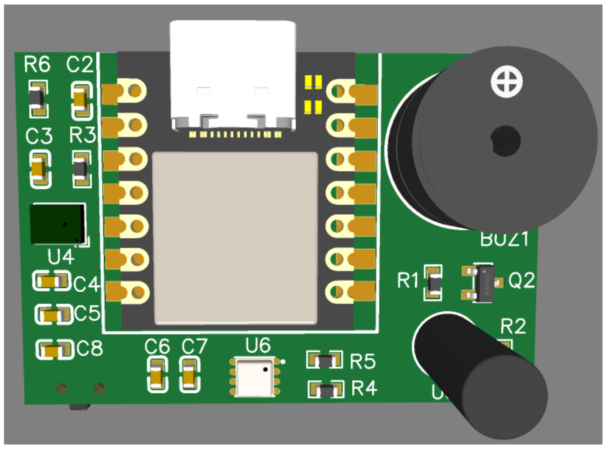
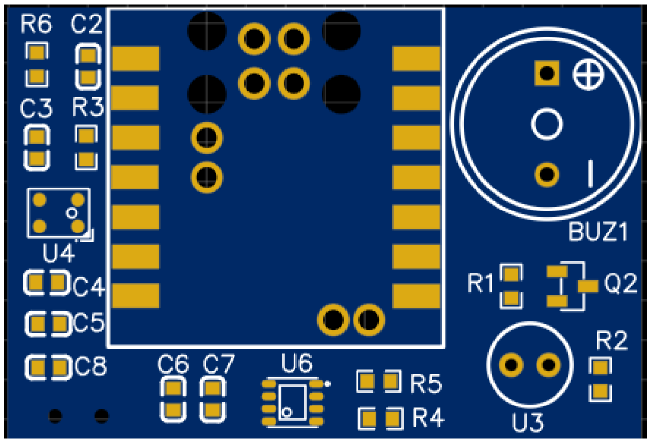

Components List
Filter by keyword
BUZ1 _ CHENZJ1:BUZZER_12X9.5RM6.5 _ B
C2 _ 10n _ C0603
C3 _ 100n _ C0603
C4 _ 100n _ C0603
C5 _ 10u _ C0603
C6 _ 0.1u _ C0603
C7 _ 0.1u _ C0603
C8 _ 100n _ C0603
Q2 _ S9013 J3 _ SOT-23-3_L2.9-W1.3-P1.90-
R1 _ 1k _ R0603
R2 _ 10K _ R0603
R3 _ 51R _ R0603
R4 _ 4.7k _ R0603
R5 _ 4.7k _ R0603
R6 _ 0K _ R0603
SW1 _ SK-3390A-L0 _ SW-SMD_SK-3390A-L
U1 _ XIAO_NRF52840 _ XIAO-NRF52840
U3 _ SW-520DS-AB12G _ SENSOR-TH_SW-
U4 _ SPU0414HR5H-SB _ MIC-SMD_4P-L3.8
U6 _ BME680 _ LGA-8_L3.0-W3.0-P0.80-TR

Figure 17

- **PCB**


R6
C2
C3
R3
U4
C4
C5
C8
C6
C7
U6
R5
R4
R1
Q2
R2
BUZ1


R6
C2
C3
R3
U4
C4
C5
C8
C6
C7
U6
R5
R4
R1
Q2
R2
BUZ1
U3