Internet of Things

1 CONTENTS

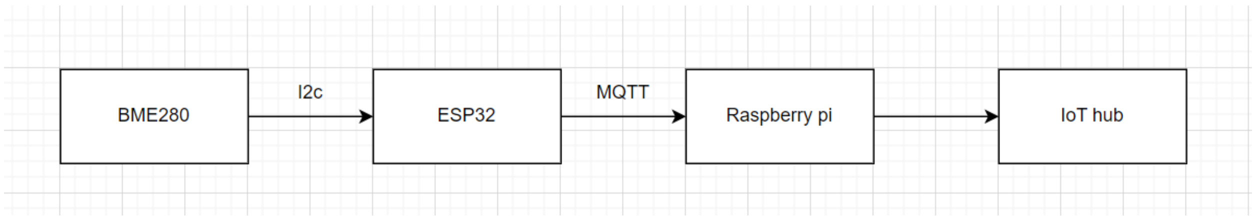
2 INTRODUCTION

The assignment is to create a small IoT network to capture the data from the bme280 sensor and store it to a cloud service as Azure. The outcome is to learn the basic of IoT (MQTT communication, I2C, …)

No	Requirements	Acknowledge
1	BME280 – ESP32	
2	ESP32 – RASPBERRY PI WIFI-NETWORK	
3	SEND DATA TO BROKER USING MQTT	
4	CREATE DATABASE AND STORE LOCALLY ON RASPBERRY PI	
5	SEND TO AZURE IOT HUB	
6	THE DATA LOSS PREVENTING PROCESS	
7	WEBSERVER VISUALISE REAL-TIME DATA	

3 DESIGN & IMPLEMENTATION

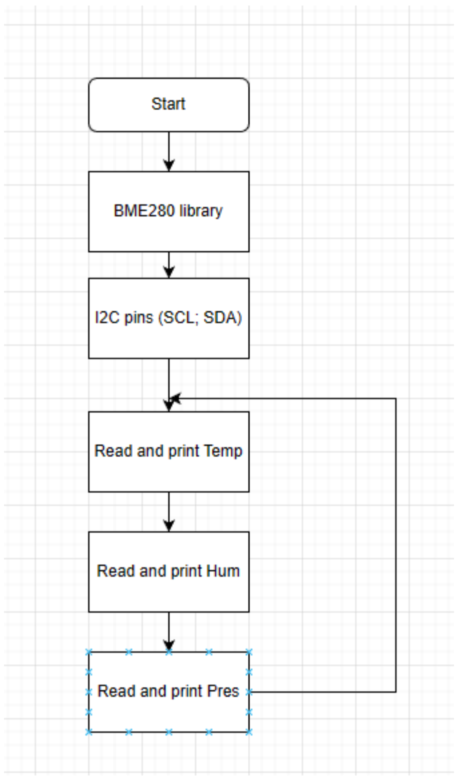
3.1 SHOW AN OVERVIEW IN A DIAGRAM OF THE COMPLETE ARCHITECTURE OF YOUR IOT SYSTEM.



1. The BME280 sensor was used to record the temperature, humidity, and pressure from the environment.
2. The ESP32 worked as an edge device which read the value from the BME280 and send the data to the gateway (raspberry pi) using MQTT.
3. The Raspberry pi is a gateway to communicate to the cloud server.
4. The IoT hub, we used Azure cloud server, included tables in the database and streaming the data.

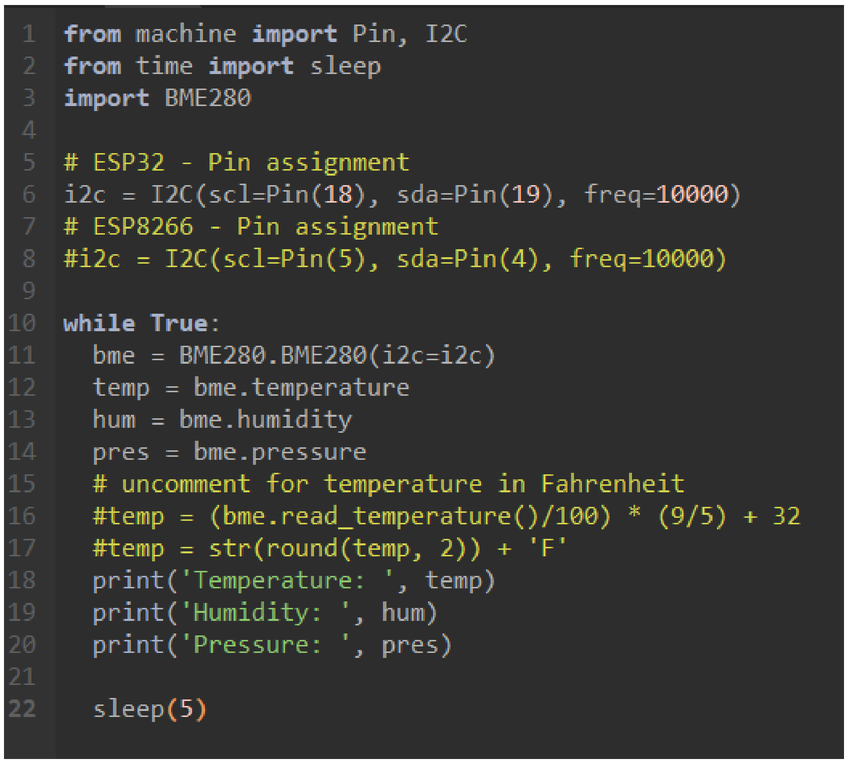
3.2 CONNECT SENSOR BME280 TO THE ESP32. COLLECT TEMPERATURE, HUMIDITY, AND PRESSURE.

Flowchart:



This is a simple Micropython program to read and print the 3 values every second. Before running the code, a BME280 library must be flashed on the ESP32 since the default library doesn't include the sensor. The library can be found on the internet (randomnerdstutorial.com).

Code:



```
from machine import Pin, I2C
from time import sleep
import BME280

# ESP32 - Pin assignment
i2c = I2C(scl=Pin(18), sda=Pin(19), freq=10000)
# ESP8266 - Pin assignment
#i2c = I2C(scl=Pin(5), sda=Pin(4), freq=10000)

while True:
  bme = BME280.BME280(i2c=i2c)
  temp = bme.temperature
  hum = bme.humidity
  pres = bme.pressure
  # uncomment for temperature in Fahrenheit
  #temp = (bme.read_temperature()/100) * (9/5) + 32
  #temp = str(round(temp, 2)) + 'F'
  print('Temperature: ', temp)
  print('Humidity: ', hum)
  print('Pressure: ', pres)

  sleep(5)
```

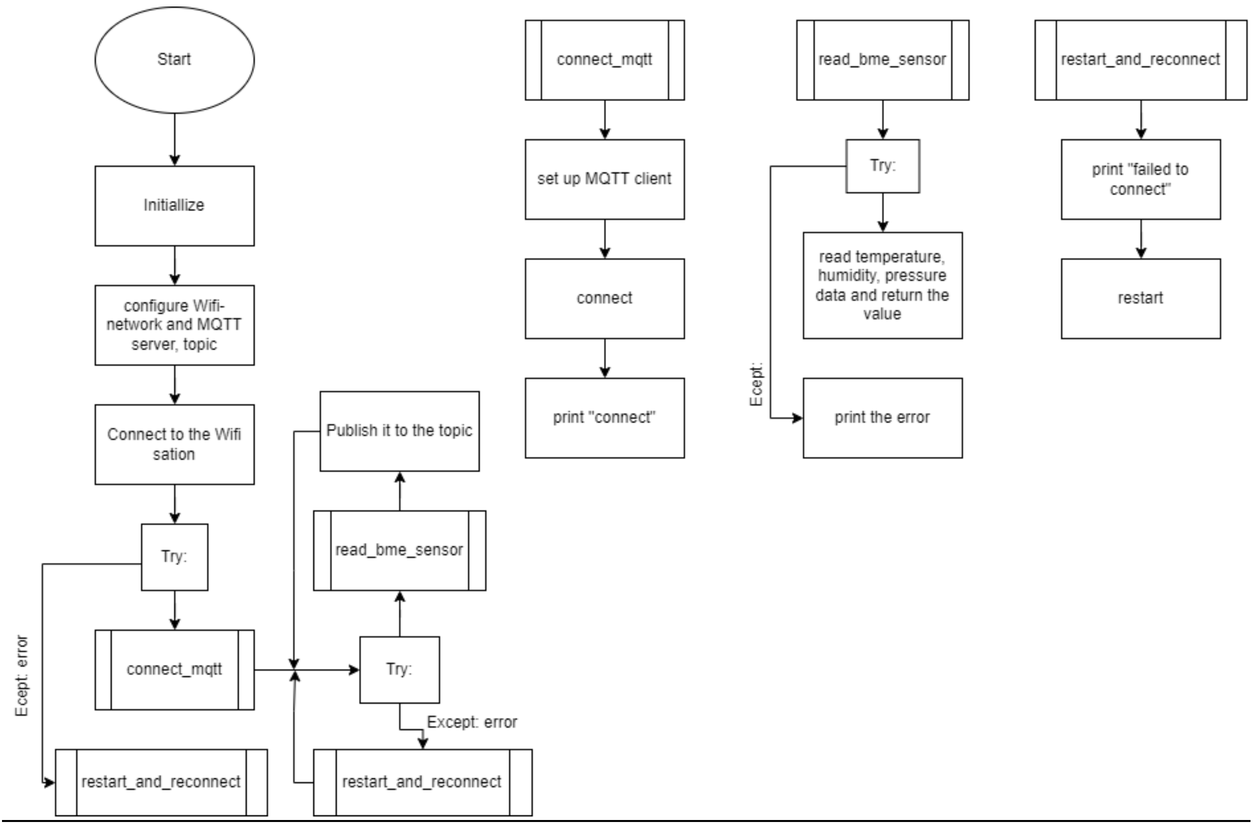
3.3 SECURITY ESP32.

Secure boot: when the ESP32 executes any software from flash, that software is trusted and signed by a known entity. If even a single bit in the software bootloader and application firmware is modified, the firmware is not trusted, and the device will refuse to execute this untrusted code. This is recommended to implement when the final program is finished. In this report, there isn't any process implementing this security. The step by step process can be found on this page: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/security/secure-boot-v1.html

Firmware secure: another way to secure the software on ESP32 is to press all the code and library into an image and flask it to the ESP32 as firmware. When the image is flask to the ESP32, the program will run automatically. The program can only be seen and modified before turning into firmware. Since micropython is not supported to implement this idea, it can't be implemented at the time of this report.

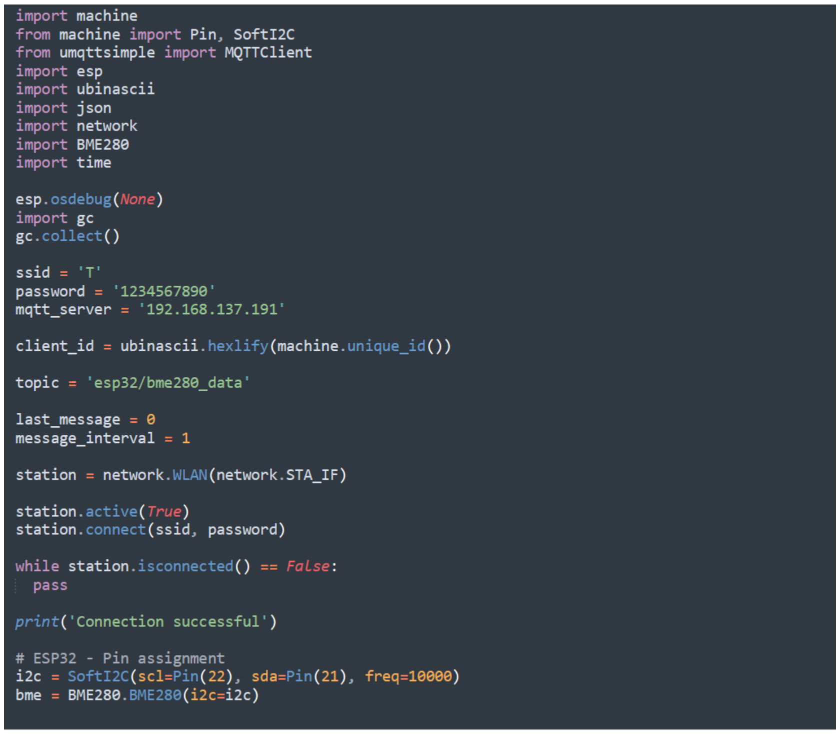
3.4 CONNECT THE ESP32 TO THE RASPBERRY PI BY A WIFI-NETWORK AND SEND THE DATA TO THE BROKER USING MQTT PROTOCOL.

ESP32 flowchart:

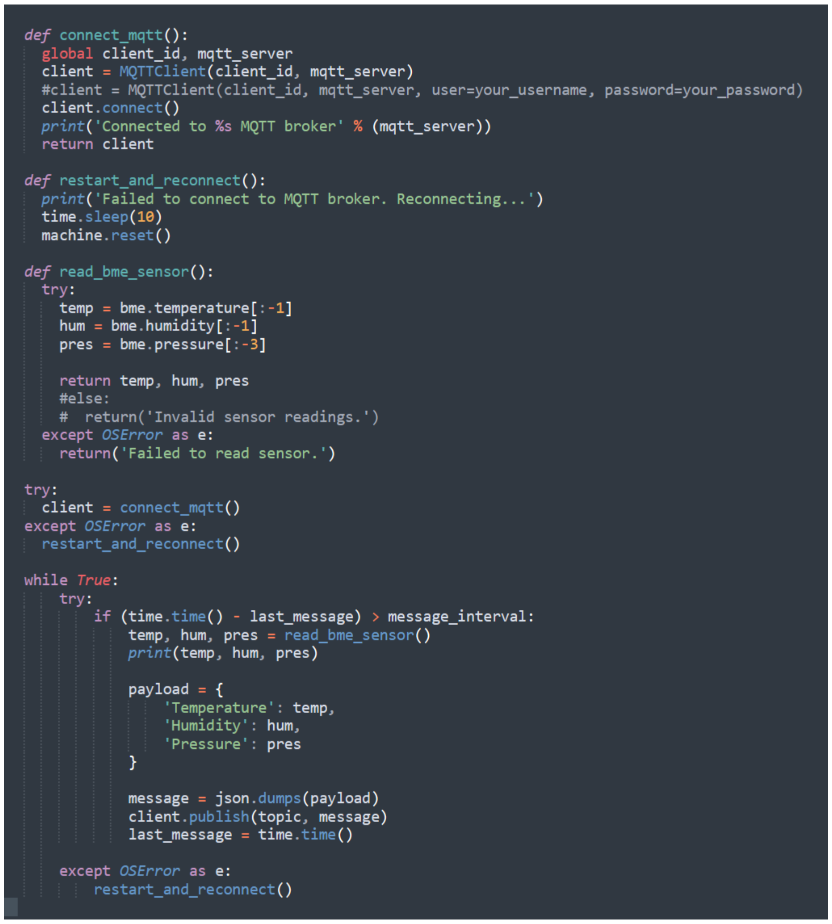


Upload umqttsimple.py library, the library can be found on the internet (randomnerdstutorial.com) to the ESP32. Check the Wi-Fi and MQTT configuration. The program will try to connect to Wi-fi and MQTT server, then read the data from bme280 sensor then publish it to the MQTT topic.

Code for ESP32:

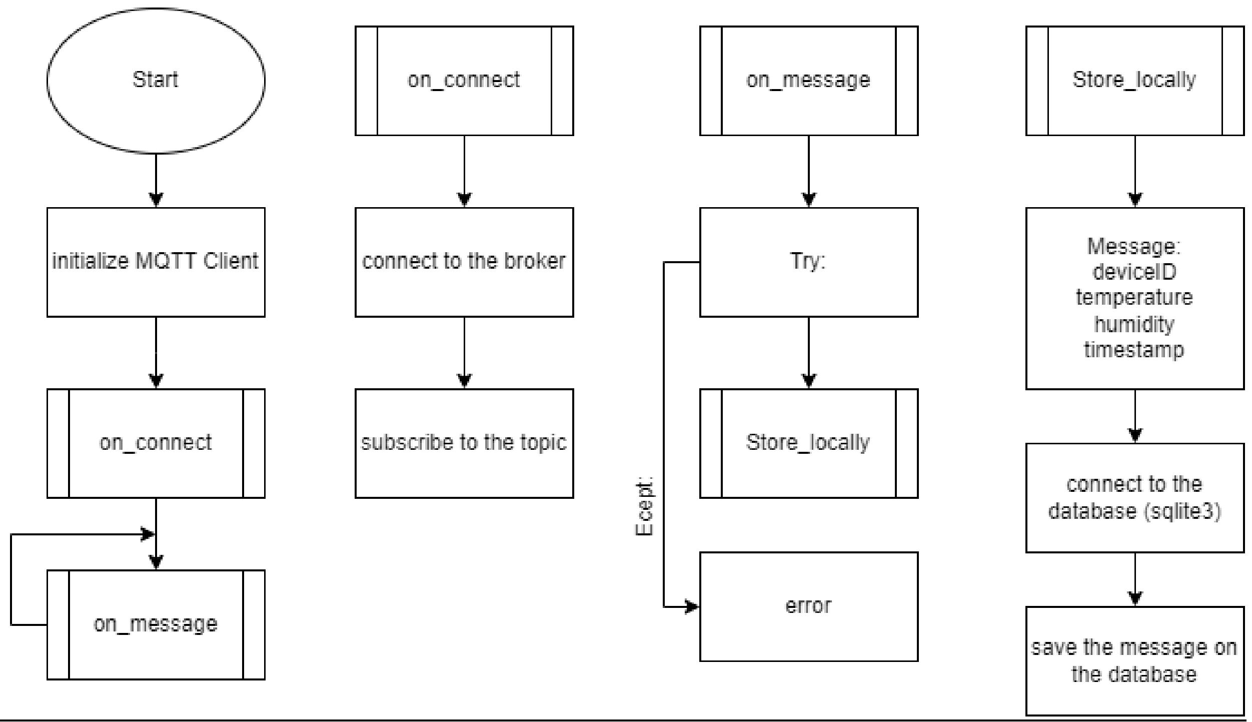


```
import machine
from machine import Pin, SoftI2C
from umqttsimple import MQTTClient
import esp
import ubinascii
import json
import network
import BME280
import time

esp.osdebug(None)
import gc
gc.collect()

ssid = 'T'
password = '1234567890'
mqtt_server = '192.168.137.191'

client_id = ubinascii.hexlify(machine.unique_id())

topic = 'esp32/bme280_data'

last_message = 0
message_interval = 1

station = network.WLAN(network.STA_IF)

station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
  pass

print('Connection successful')

# ESP32 - Pin assignment
i2c = SoftI2C(scl=Pin(22), sda=Pin(21), freq=10000)
bme = BME280.BME280(i2c=i2c)
```



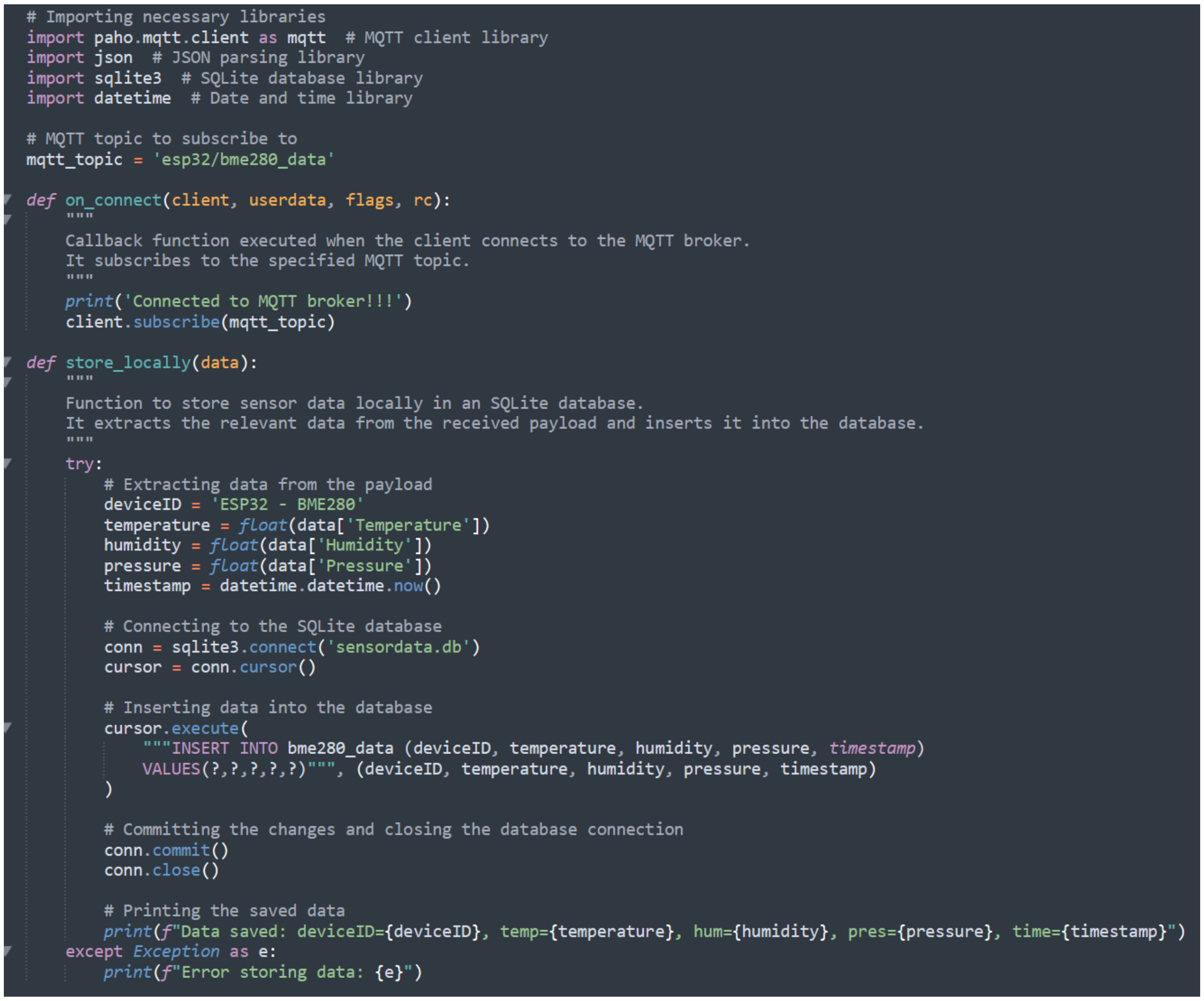
```
def connect_mqtt():
  global client_id, mqtt_server
  client = MQTTClient(client_id, mqtt_server)
  #client = MQTTClient(client_id, mqtt_server, user=your_username, password=your_password)
  client.connect()
  print('Connected to %s MQTT broker' % (mqtt_server))
  return client

def restart_and_reconnect():
  print('Failed to connect to MQTT broker. Reconnecting...')
  time.sleep(10)
  machine.reset()

def read_bme_sensor():
  try:
    temp = bme.temperature[:-1]
    hum = bme.humidity[:-1]
    pres = bme.pressure[:-3]

    return temp, hum, pres
    #else:
    #  return('Invalid sensor readings.')
  except OSError as e:
    return('Failed to read sensor.')

try:
  client = connect_mqtt()
except OSError as e:
  restart_and_reconnect()

while True:
    try:
        if (time.time() - last_message) > message_interval:
            temp, hum, pres = read_bme_sensor()
            print(temp, hum, pres)

            payload = {
                'Temperature': temp,
                'Humidity': hum,
                'Pressure': pres
            }

            message = json.dumps(payload)
            client.publish(topic, message)
            last_message = time.time()

    except OSError as e:
        restart_and_reconnect()
```

Raspberry Pi flowchart:

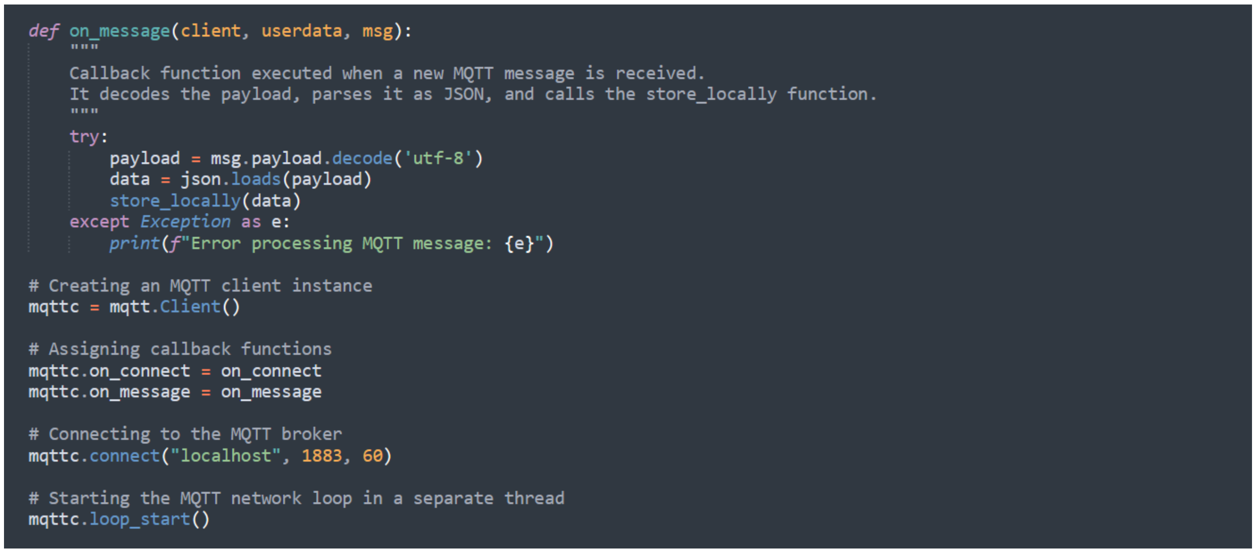


Remember to run "mosquitto -d" in the terminal before running the program. The program will connect the MQTT server and take the message from the topic to store it to the database on raspberry pi.

Code for Raspberry Pi:



```python
# Importing necessary libraries
import paho.mqtt.client as mqtt  # MQTT client library
import json  # JSON parsing library
import sqlite3  # SQLite database library
import datetime  # Date and time library

# MQTT topic to subscribe to
mqtt_topic = 'esp32/bme280_data'

def on_connect(client, userdata, flags, rc):
    """
    Callback function executed when the client connects to the MQTT broker.
    It subscribes to the specified MQTT topic.
    """
    print('Connected to MQTT broker!!!')
    client.subscribe(mqtt_topic)

def store_locally(data):
    """
    Function to store sensor data locally in an SQLite database.
    It extracts the relevant data from the received payload and inserts it into the database.
    """
    try:
        # Extracting data from the payload
        deviceID = 'ESP32 - BME280'
        temperature = float(data['Temperature'])
        humidity = float(data['Humidity'])
        pressure = float(data['Pressure'])
        timestamp = datetime.datetime.now()

        # Connecting to the SQLite database
        conn = sqlite3.connect('sensordata.db')
        cursor = conn.cursor()

        # Inserting data into the database
        cursor.execute(
            """INSERT INTO bme280_data (deviceID, temperature, humidity, pressure, timestamp)
            VALUES(?,?,?,?,?)""", (deviceID, temperature, humidity, pressure, timestamp)
        )

        # Committing the changes and closing the database connection
        conn.commit()
        conn.close()

        # Printing the saved data
        print(f"Data saved: deviceID={deviceID}, temp={temperature}, hum={humidity}, pres={pressure}, time={timestamp}")
    except Exception as e:
        print(f"Error storing data: {e}")
```



```python
def on_message(client, userdata, msg):
    """
    Callback function executed when a new MQTT message is received.
    It decodes the payload, parses it as JSON, and calls the store_locally function.
    """
    try:
        payload = msg.payload.decode('utf-8')
        data = json.loads(payload)
        store_locally(data)
    except Exception as e:
        print(f"Error processing MQTT message: {e}")

# Creating an MQTT client instance
mqttc = mqtt.Client()

# Assigning callback functions
mqttc.on_connect = on_connect
mqttc.on_message = on_message

# Connecting to the MQTT broker
mqttc.connect("localhost", 1883, 60)

# Starting the MQTT network loop in a separate thread
mqttc.loop_start()
```

3.5 SECURE THE MQTT SERVER AND THE DATA STREAM TO THE SERVER.

To secure the MQTT server, password authentication is implemented as installing the mosquitto on raspberry pi. Create a username and password then save on passwd file on mosquitto directory, adjust the configuration on the conf file to activate the function. More details about which function or how to do it step by step can be found on randomnerdtutorials. The code in the program on both esp32 and raspberry pi should also be adjusted to work with the authentication system, the final code can be seen in section 7.

Link: https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/

Secure Sockets Layer (SSL) provides data encryption, data integrity and authentication, which means that when using SSL, no one has read the message, no one has changed the message. The content is encrypted (unreadable for human beings). The idea for implementing is to create 2 different ports: internal and external ports. The local port can only access locally with the private key, so it is not accessible externally. The encrypted listener is set up on external port (the standard port for MQTT + SSL, also known as MQTTS). The external access via this port can only seen encrypted messages, so the message won't be revealed or changed by other people. The detail implementing can be found on this page: https://www.digitalocean.com/community/tutorials/how-to-install-and-secure-the-mosquitto-mqtt-messaging-broker-on-debian-10

3.6 CREATE A DATABASE ON THE RASPBERRY PI TO STORE DATA LOCALLY.

Create the database using SQLite on raspberry pi on terminal: "sqlite3 sensordata.db" after going to the sensordata.db, create a table name bme280_data with these functions:

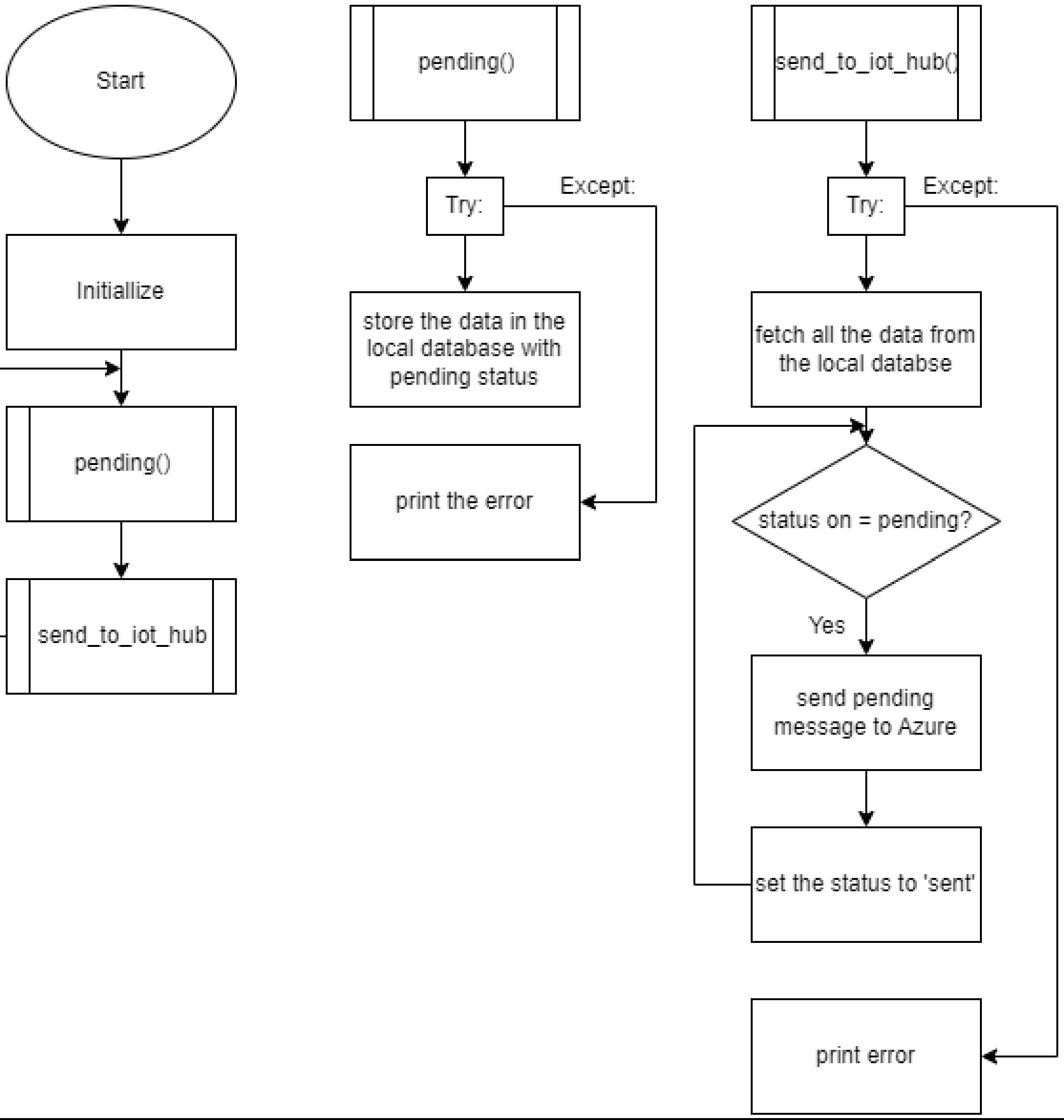
BEGIN;

CREATE TABLE bme280_data (deviceID TEXT, temperature REAL, humidity REAL, pressure REAL, timestamp DATETIME);

COMMIT;

The code for this part is already included in 5&6. After creating the table, running mosquitto in the background, run the esp32 code to start sending the data to the MQTT server, then run the Raspberry pi code to receive and store the data in the database.

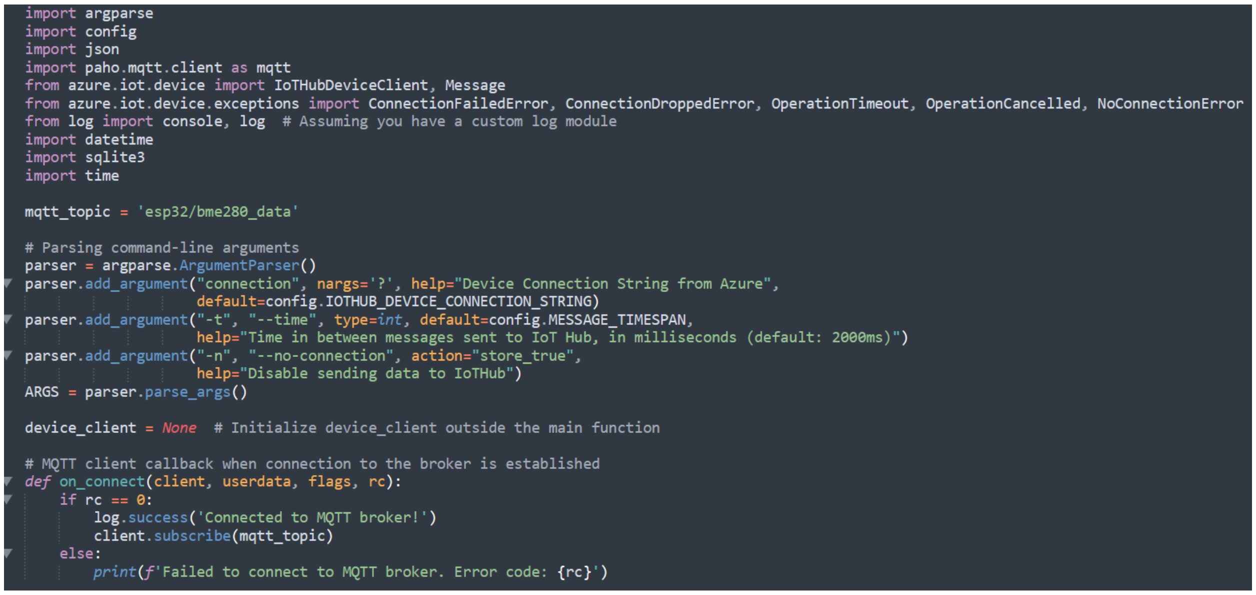
3.7 CONNECT THE RASPBERRY PI TO THE IOT HUB.



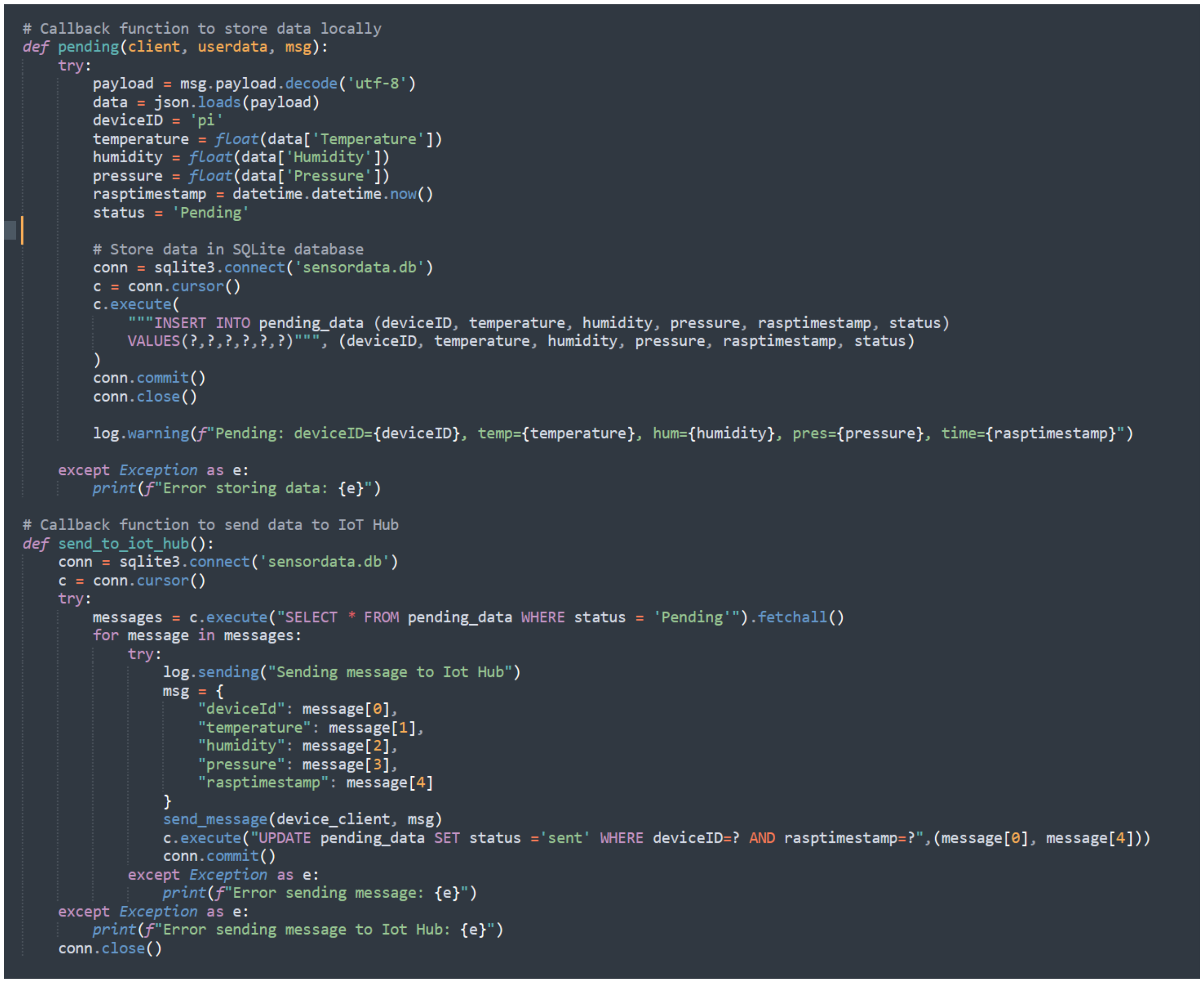
The code on ESP32 is the same as above. The above flowchart is for the updated app.py code for raspberry pi.

Design: after initializing, the raspberry pi will receive the data transferred from the ESP32 via MQTT and send it to the local database as pending data. Then, if the connection is successful to the IoT Hub, the program will prioritize sending all the pending data in the database before sending the new data. In meantime, the data from esp32 is still being sent and stored as pending data in the database in order to prevent losses.

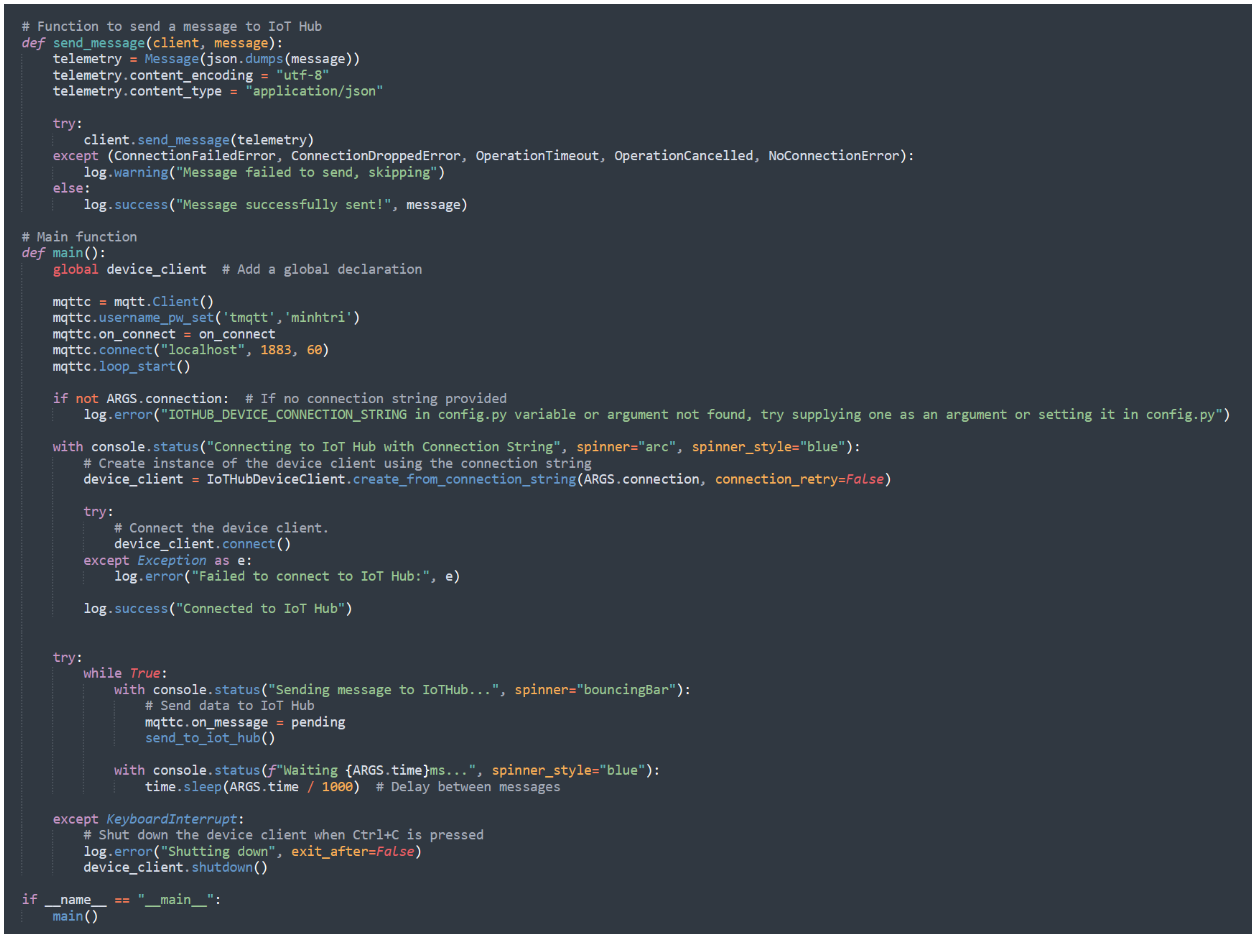
Code:



```python
import argparse
import config
import json
import paho.mqtt.client as mqtt
from azure.iot.device import IoTHubDeviceClient, Message
from azure.iot.device.exceptions import ConnectionFailedError, ConnectionDroppedError, OperationTimeout, OperationCancelled, NoConnectionError
from log import console, log  # Assuming you have a custom log module
import datetime
import sqlite3
import time

mqtt_topic = 'esp32/bme280_data'

# Parsing command-line arguments
parser = argparse.ArgumentParser()
parser.add_argument("connection", nargs='?', help="Device Connection String from Azure",
                    default=config.IOTHUB_DEVICE_CONNECTION_STRING)
parser.add_argument("-t", "--time", type=int, default=config.MESSAGE_TIMESPAN,
                    help="Time in between messages sent to IoT Hub, in milliseconds (default: 2000ms)")
parser.add_argument("-n", "--no-connection", action="store_true",
                    help="Disable sending data to IoTHub")
ARGS = parser.parse_args()

device_client = None  # Initialize device_client outside the main function

# MQTT client callback when connection to the broker is established
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        log.success('Connected to MQTT broker!')
        client.subscribe(mqtt_topic)
    else:
        print(f'Failed to connect to MQTT broker. Error code: {rc}')
```



```python
# Callback function to store data locally
def pending(client, userdata, msg):
    try:
        payload = msg.payload.decode('utf-8')
        data = json.loads(payload)
        deviceID = 'pi'
        temperature = float(data['Temperature'])
        humidity = float(data['Humidity'])
        pressure = float(data['Pressure'])
        rasptimestamp = datetime.datetime.now()
        status = 'Pending'

        # Store data in SQLite database
        conn = sqlite3.connect('sensordata.db')
        c = conn.cursor()
        c.execute(
            """INSERT INTO pending_data (deviceID, temperature, humidity, pressure, rasptimestamp, status)
            VALUES(?,?,?,?,?,?)""", (deviceID, temperature, humidity, pressure, rasptimestamp, status)
        )
        conn.commit()
        conn.close()

        log.warning(f"Pending: deviceID={deviceID}, temp={temperature}, hum={humidity}, pres={pressure}, time={rasptimestamp}")

    except Exception as e:
        print(f"Error storing data: {e}")

# Callback function to send data to IoT Hub
def send_to_iot_hub():
    conn = sqlite3.connect('sensordata.db')
    c = conn.cursor()
    try:
        messages = c.execute("SELECT * FROM pending_data WHERE status = 'Pending'").fetchall()
        for message in messages:
            try:
                log.sending("Sending message to Iot Hub")
                msg = {
                    "deviceId": message[0],
                    "temperature": message[1],
                    "humidity": message[2],
                    "pressure": message[3],
                    "rasptimestamp": message[4]
                }
                send_message(device_client, msg)
                c.execute("UPDATE pending_data SET status ='sent' WHERE deviceID=? AND rasptimestamp=?",(message[0], message[4]))
                conn.commit()
            except Exception as e:
                print(f"Error sending message: {e}")
    except Exception as e:
        print(f"Error sending message to Iot Hub: {e}")
    conn.close()
```



```python
# Function to send a message to IoT Hub
def send_message(client, message):
    telemetry = Message(json.dumps(message))
    telemetry.content_encoding = "utf-8"
    telemetry.content_type = "application/json"

    try:
        client.send_message(telemetry)
    except (ConnectionFailedError, ConnectionDroppedError, OperationTimeout, OperationCancelled, NoConnectionError):
        log.warning("Message failed to send, skipping")
    else:
        log.success("Message successfully sent!", message)

# Main function
def main():
    global device_client  # Add a global declaration

    mqttc = mqtt.Client()
    mqttc.username_pw_set('tmqtt','minhtri')
    mqttc.on_connect = on_connect
    mqttc.connect("localhost", 1883, 60)
    mqttc.loop_start()

    if not ARGS.connection:  # If no connection string provided
        log.error("IOTHUB_DEVICE_CONNECTION_STRING in config.py variable or argument not found, try supplying one as an argument or setting it in config.py")

    with console.status("Connecting to IoT Hub with Connection String", spinner="arc", spinner_style="blue"):
        # Create instance of the device client using the connection string
        device_client = IoTHubDeviceClient.create_from_connection_string(ARGS.connection, connection_retry=False)

        try:
            # Connect the device client.
            device_client.connect()
        except Exception as e:
            log.error("Failed to connect to IoT Hub:", e)

        log.success("Connected to IoT Hub")


    try:
        while True:
            with console.status("Sending message to IoTHub...", spinner="bouncingBar"):
                # Send data to IoT Hub
                mqttc.on_message = pending
                send_to_iot_hub()

            with console.status(f"Waiting {ARGS.time}ms...", spinner_style="blue"):
                time.sleep(ARGS.time / 1000)  # Delay between messages

    except KeyboardInterrupt:
        # Shut down the device client when Ctrl+C is pressed
        log.error("Shutting down", exit_after=False)
        device_client.shutdown()

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

3.8 WEBSERVER

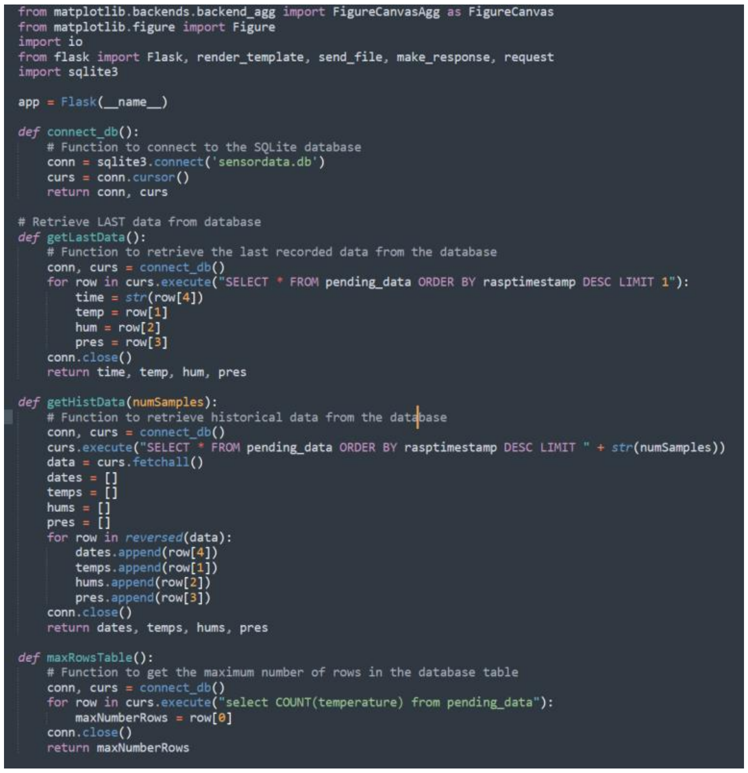
Main (webserver.py) code on raspberry pi. The idea of the webserver is to take the latest value from the local database and make 3 gages (temp, humi, pres) and 3 graphs drew by taking the number of datapoint in the database. The number can be changed directly on the website by the user. To implement the program, we need 2 more folders in the same directory as the webserver.py called templates and static as below:

Webserver.py

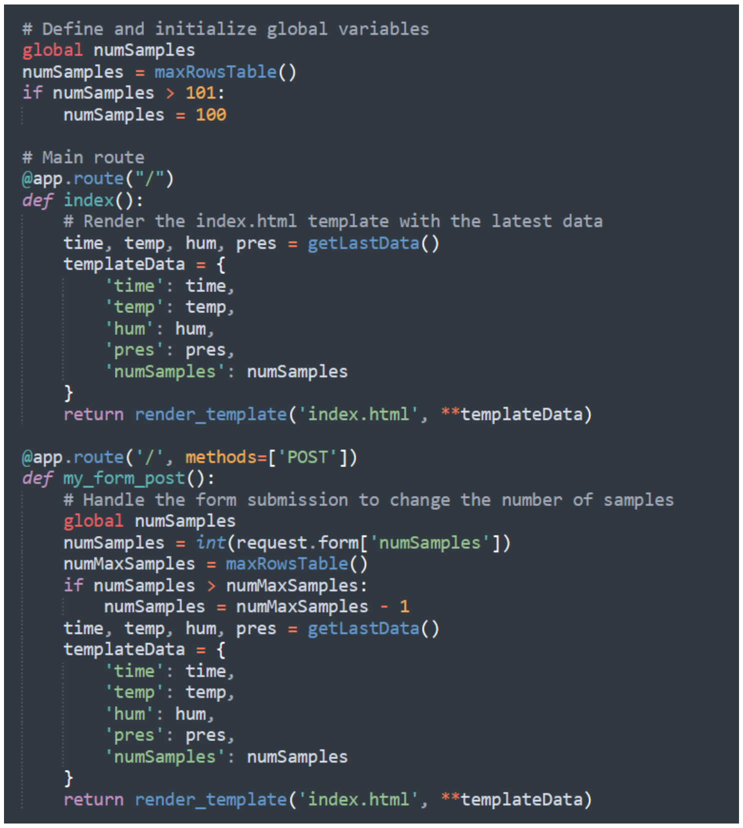
Templates (folder): index.html

Static (folder): style.css; justgage.js; raphael-2.1.4.min.js

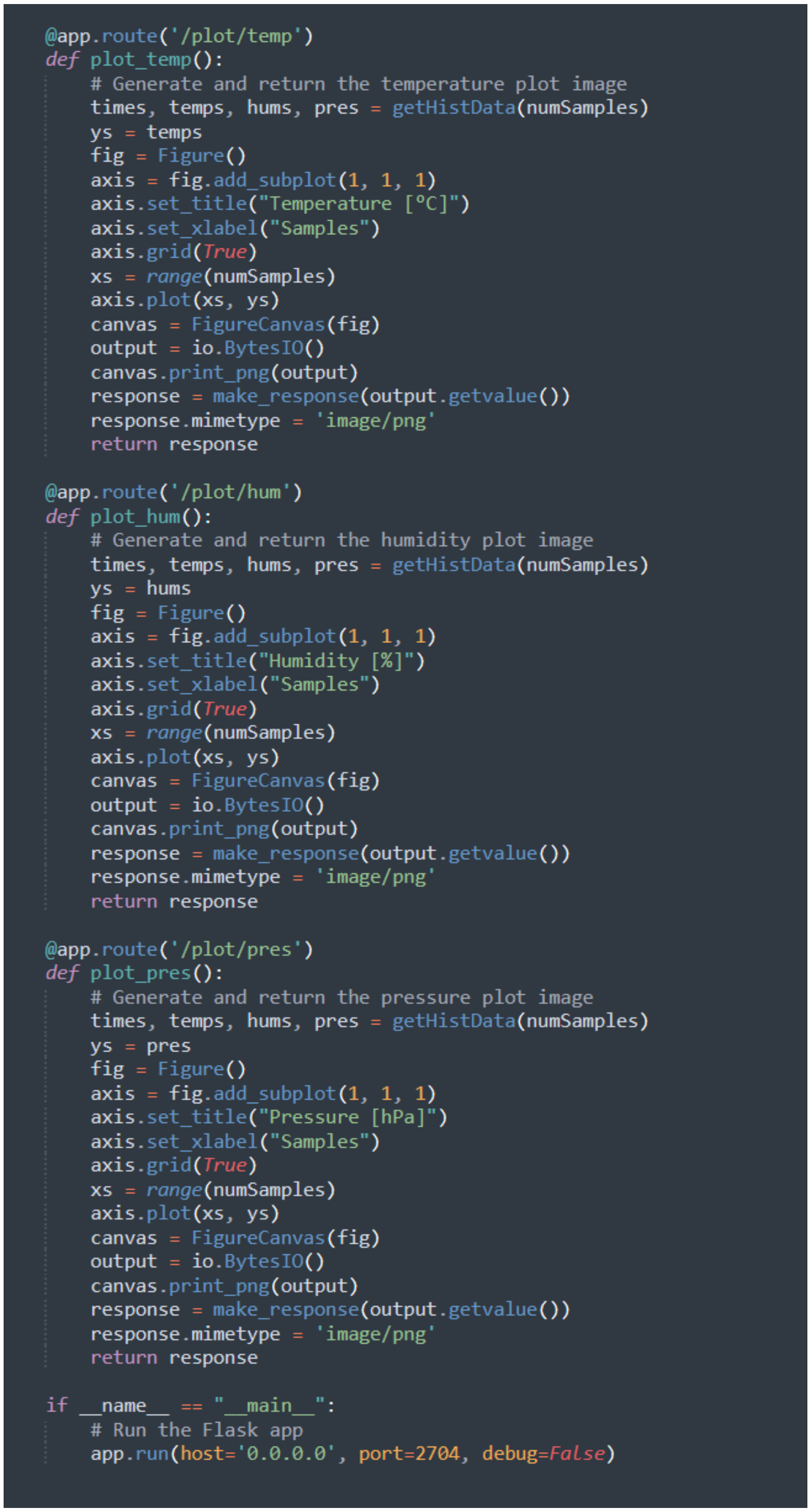
Code for webserver.py



```python
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import io
from flask import Flask, render_template, send_file, make_response, request
import sqlite3

app = Flask(__name__)

def connect_db():
    # Function to connect to the SQLite database
    conn = sqlite3.connect('sensordata.db')
    curs = conn.cursor()
    return conn, curs

# Retrieve LAST data from database
def getLastData():
    # Function to retrieve the last recorded data from the database
    conn, curs = connect_db()
    for row in curs.execute("SELECT * FROM pending_data ORDER BY rasptimestamp DESC LIMIT 1"):
        time = str(row[4])
        temp = row[1]
        hum = row[2]
        pres = row[3]
    conn.close()
    return time, temp, hum, pres

def getHistData(numSamples):
    # Function to retrieve historical data from the database
    conn, curs = connect_db()
    curs.execute("SELECT * FROM pending_data ORDER BY rasptimestamp DESC LIMIT " + str(numSamples))
    data = curs.fetchall()
    dates = []
    temps = []
    hums = []
    pres = []
    for row in reversed(data):
        dates.append(row[4])
        temps.append(row[1])
        hums.append(row[2])
        pres.append(row[3])
    conn.close()
    return dates, temps, hums, pres

def maxRowsTable():
    # Function to get the maximum number of rows in the database table
    conn, curs = connect_db()
    for row in curs.execute("select COUNT(temperature) from pending_data"):
        maxNumberRows = row[0]
    conn.close()
    return maxNumberRows
```



```python
# Define and initialize global variables
global numSamples
numSamples = maxRowsTable()
if numSamples > 101:
    numSamples = 100

# Main route
@app.route("/")
def index():
    # Render the index.html template with the latest data
    time, temp, hum, pres = getLastData()
    templateData = {
        'time': time,
        'temp': temp,
        'hum': hum,
        'pres': pres,
        'numSamples': numSamples
    }
    return render_template('index.html', **templateData)

@app.route('/', methods=['POST'])
def my_form_post():
    # Handle the form submission to change the number of samples
    global numSamples
    numSamples = int(request.form['numSamples'])
    numMaxSamples = maxRowsTable()
    if numSamples > numMaxSamples:
        numSamples = numMaxSamples - 1
    time, temp, hum, pres = getLastData()
    templateData = {
        'time': time,
        'temp': temp,
        'hum': hum,
        'pres': pres,
        'numSamples': numSamples
    }
    return render_template('index.html', **templateData)
```



```python
@app.route('/plot/temp')
def plot_temp():
    # Generate and return the temperature plot image
    times, temps, hums, pres = getHistData(numSamples)
    ys = temps
    fig = Figure()
    axis = fig.add_subplot(1, 1, 1)
    axis.set_title("Temperature [°C]")
    axis.set_xlabel("Samples")
    axis.grid(True)
    xs = range(numSamples)
    axis.plot(xs, ys)
    canvas = FigureCanvas(fig)
    output = io.BytesIO()
    canvas.print_png(output)
    response = make_response(output.getvalue())
    response.mimetype = 'image/png'
    return response

@app.route('/plot/hum')
def plot_hum():
    # Generate and return the humidity plot image
    times, temps, hums, pres = getHistData(numSamples)
    ys = hums
    fig = Figure()
    axis = fig.add_subplot(1, 1, 1)
    axis.set_title("Humidity [%]")
    axis.set_xlabel("Samples")
    axis.grid(True)
    xs = range(numSamples)
    axis.plot(xs, ys)
    canvas = FigureCanvas(fig)
    output = io.BytesIO()
    canvas.print_png(output)
    response = make_response(output.getvalue())
    response.mimetype = 'image/png'
    return response

@app.route('/plot/pres')
def plot_pres():
    # Generate and return the pressure plot image
    times, temps, hums, pres = getHistData(numSamples)
    ys = pres
    fig = Figure()
    axis = fig.add_subplot(1, 1, 1)
    axis.set_title("Pressure [hPa]")
    axis.set_xlabel("Samples")
    axis.grid(True)
    xs = range(numSamples)
    axis.plot(xs, ys)
    canvas = FigureCanvas(fig)
    output = io.BytesIO()
    canvas.print_png(output)
    response = make_response(output.getvalue())
    response.mimetype = 'image/png'
    return response

if __name__ == "__main__":
    # Run the Flask app
    app.run(host='0.0.0.0', port=2704, debug=False)
```

Code for index.html:

```
<!doctype html>
<html>

<head>
  <title>IoT assignment</title>
  <link rel="stylesheet" href='../static/style.css'/>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <style>
  body {
    text-align: center;
    }

    .gauge-container {
      display: flex;
      justify-content: center;
      margin: 1em;
    }

    .gauge {
      width: 200px;
      height: 160px;
      margin: 0 1em;
    }

    .plot-container {
      display: flex;
      flex-wrap: wrap;
      justify-content: center;
      margin: 1em;
    }

    .plot {
      width: 49%;
      margin: 1em;
    }
  </style>
</head>

<body>
  <h1>BME280 sensor Data </h1>
  <div class="gauge-container">
    <div id="g1" class="gauge"></div>
    <div id="g2" class="gauge"></div>
    <div id="g3" class="gauge"></div>
</div>
<hr>
<h3> Last Sensors Reading: {{ time }} ==> <a href="/" class="button">REFRESH</a></h3>
<hr>
<h3> HISTORICAL DATA </h3>
    <p> Enter number of samples to retrieve:
    <form method="POST">
        <input name="numSamples" value="{{numSamples}}">
        <input type="submit">
    </form></p>
    <hr>
<div class="plot-container">
    <div class="plot">
        <img src="/plot/temp" alt="Temperature Plot">
    </div>
    <div class="plot">
        <img src="/plot/hum" alt="Humidity Plot">
    </div>
    <div class="plot">
        <img src="/plot/pres" alt="Pressure Plot">
    </div>
</div>

<p> @2018 Developed by MJRoBot.org</p>

<script src="../static/raphael-2.1.4.min.js"></script>
<script src="../static/justgage.js"></script>
<script>
    var g1, g2, g3;
    document.addEventListener("DOMContentLoaded", function(event) {
        g1 = new JustGage({
            id: "g1",
            value: {{temp}},
            valueFontColor: "yellow",
            titleFontColor: "yellow",
            min: -10,
            max: 50,
            title: "Temperature",
            label: "Celcius"
        });

        g2 = new JustGage({
            id: "g2",
            value: {{hum}},
            valueFontColor: "yellow",
            titleFontColor: "yellow",
            min: 0,
            max: 100,
            title: "Humidity",
            label: "%"
        });
        g3 = new JustGage({
            id: "g3",
            value: {{pres}},
            valueFontColor: "yellow",
            titleFontColor: "yellow",
            min: 0,
            max: 10000,
            title: "Pressure",
            label: "hPa"
        });
    });
  </script>
</body>

</html>
```

Code for style.css:

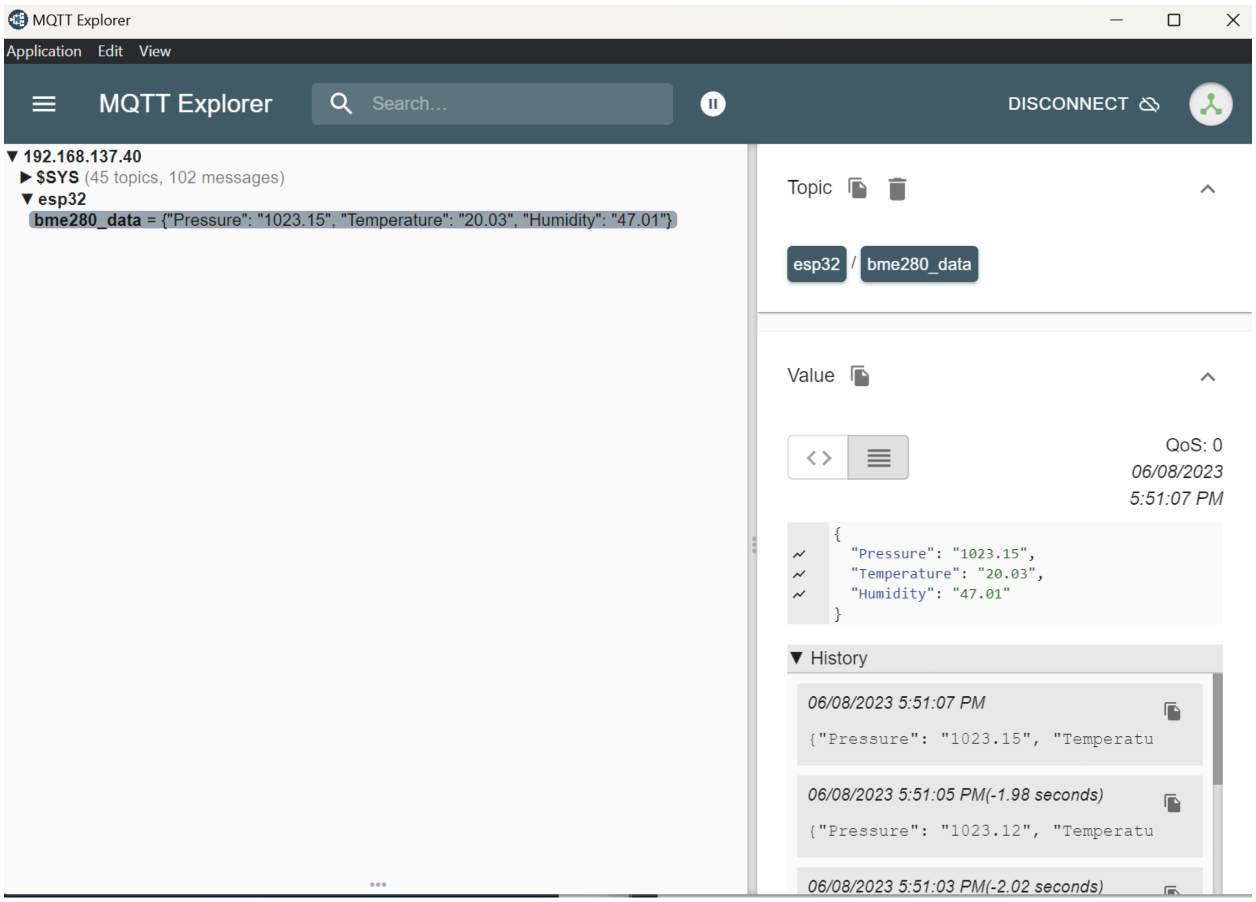
```
body{
  background: #414a4c;
  color: white;
  padding:1%
}

.button {
  font: bold 15px Arial;
  text-decoration: none;
  background-color: #EEEEEE;
  color: #333333;
  padding: 2px 6px 2px 6px;
  border-top: 1px solid #CCCCCC;
  border-right: 1px solid #333333;
  border-bottom: 1px solid #333333;
  border-left: 1px solid #CCCCCC;
}
img{
  display: display: inline-block
}
```

4 RESULT

Reading the data from bme280 on esp32 and send to the MQTT broker.

```
Connection successful
Connected to 192.168.137.40 MQTT broker
20.24 46.81 1022.75
20.05 46.88 1023.13
20.05 46.87 1023.13
20.04 46.87 1023.21
20.05 46.87 1023.18
20.05 46.87 1023.15
20.05 46.85 1023.13
20.04 46.87 1023.13
```

Use MQTT explorer to check the data sent to the broker.



```
▼ 192.168.137.40
  ▶ $SYS (45 topics, 102 messages)
  ▼ esp32
    bme280_data = {"Pressure": "1023.15", "Temperature": "20.03", "Humidity": "47.01"}
```

esp32 / bme280_data

QoS: 0
06/08/2023
5:51:07 PM

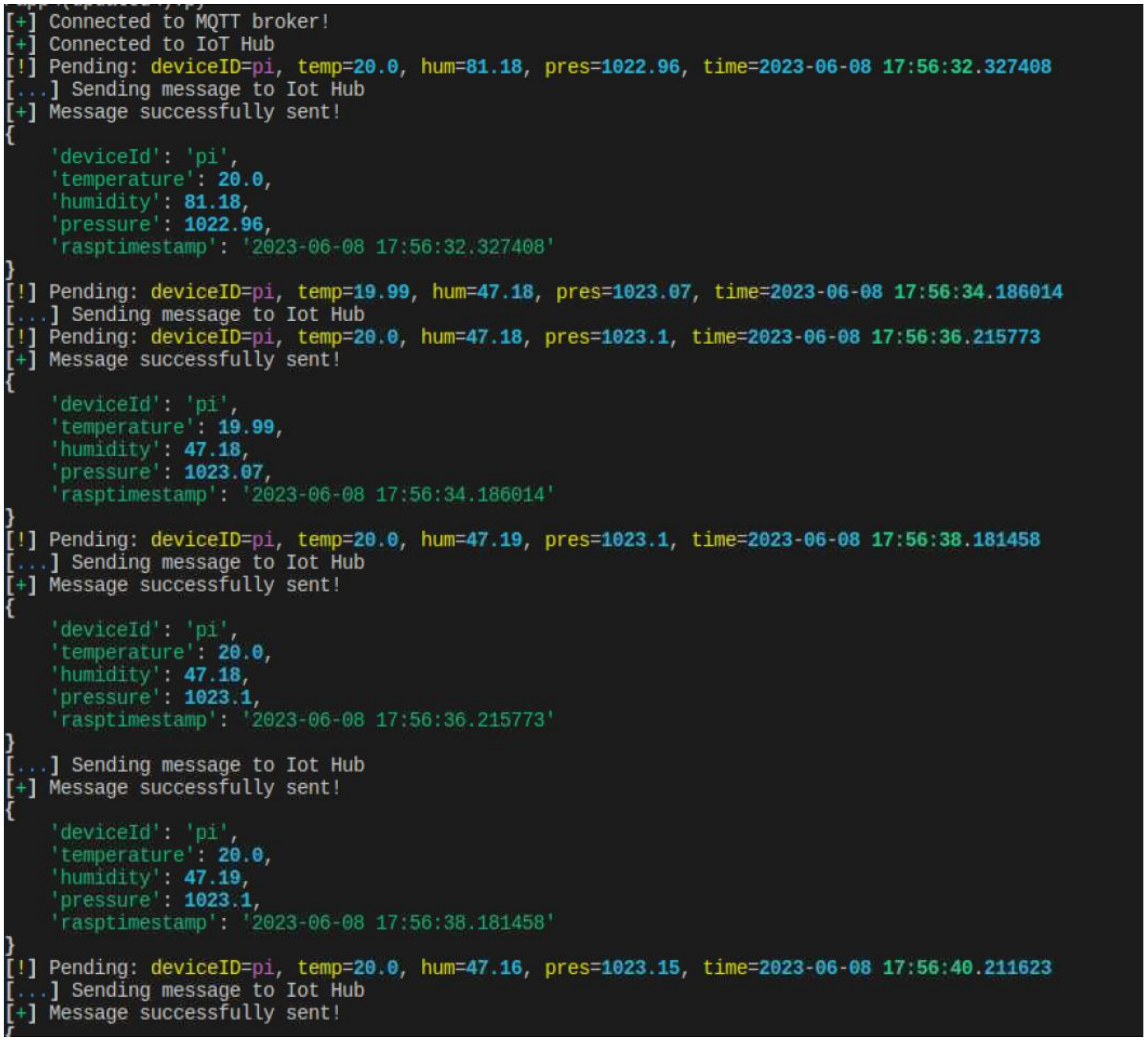
```
{
  "Pressure": "1023.15",
  "Temperature": "20.03",
  "Humidity": "47.01"
}
```

06/08/2023 5:51:07 PM
{"Pressure": "1023.15", "Temperatu

06/08/2023 5:51:05 PM(-1.98 seconds)
{"Pressure": "1023.12", "Temperatu

06/08/2023 5:51:03 PM(-2.02 seconds)

Now we can clearly see that the data sent to the broker, we need to check the data received from the raspberry pi.



```
[+] Connected to MQTT broker!
[+] Connected to IoT Hub
[!] Pending: deviceID=pi, temp=20.0, hum=81.18, pres=1022.96, time=2023-06-08 17:56:32.327408
[...] Sending message to Iot Hub
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 20.0,
    'humidity': 81.18,
    'pressure': 1022.96,
    'rasptimestamp': '2023-06-08 17:56:32.327408'
}
[!] Pending: deviceID=pi, temp=19.99, hum=47.18, pres=1023.07, time=2023-06-08 17:56:34.186014
[...] Sending message to Iot Hub
[!] Pending: deviceID=pi, temp=20.0, hum=47.18, pres=1023.1, time=2023-06-08 17:56:36.215773
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 19.99,
    'humidity': 47.18,
    'pressure': 1023.07,
    'rasptimestamp': '2023-06-08 17:56:34.186014'
}
[!] Pending: deviceID=pi, temp=20.0, hum=47.19, pres=1023.1, time=2023-06-08 17:56:38.181458
[...] Sending message to Iot Hub
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 20.0,
    'humidity': 47.18,
    'pressure': 1023.1,
    'rasptimestamp': '2023-06-08 17:56:36.215773'
}
[...] Sending message to Iot Hub
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 20.0,
    'humidity': 47.19,
    'pressure': 1023.1,
    'rasptimestamp': '2023-06-08 17:56:38.181458'
}
[!] Pending: deviceID=pi, temp=20.0, hum=47.16, pres=1023.15, time=2023-06-08 17:56:40.211623
[...] Sending message to Iot Hub
[+] Message successfully sent!
```

The data is stored in the local database as pending then sent to Azure. The delay between each message is 2 seconds. We can also see that there is a moment at 17:56:34, it failed to send the message to the Iot hub but the program still received the data from the esp32 via MQTT at 17:56:36. When the system back to online, it prioritized sending pending message before the new one. The implementation was successfully showed the data loss preventing process.

Data stored in pending_data database:

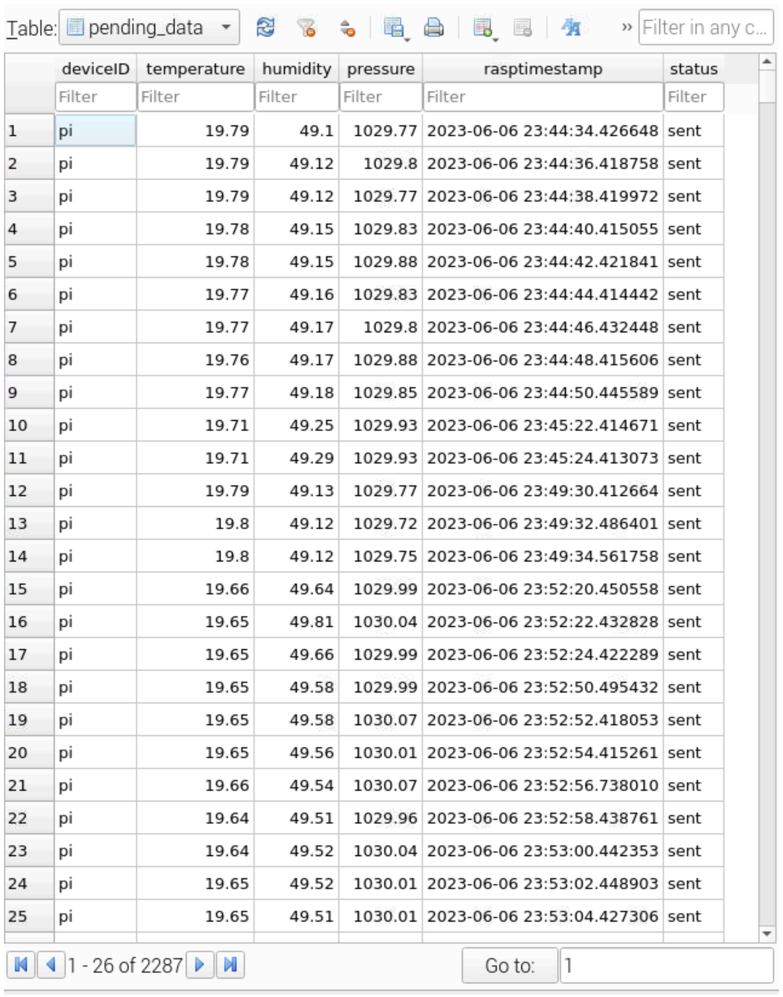
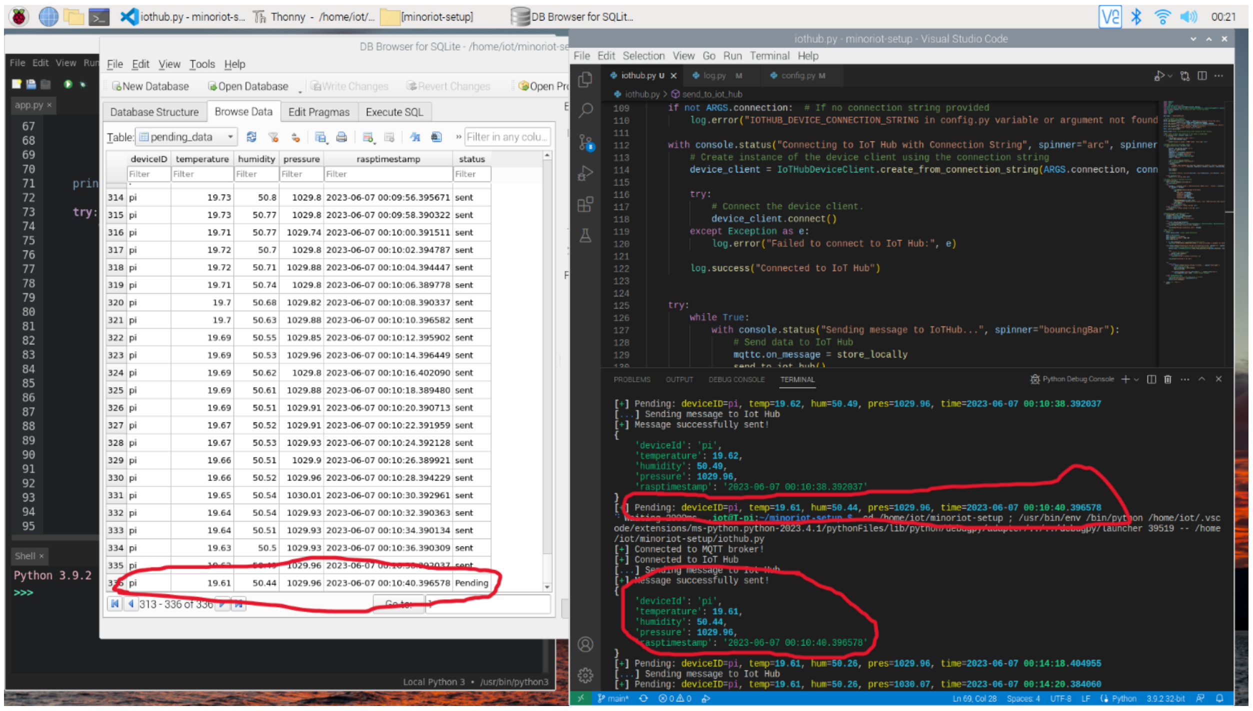


Table: pending_data

	deviceID	temperature	humidity	pressure	rasptimestamp	status
1	pi	19.79	49.1	1029.77	2023-06-06 23:44:34.426648	sent
2	pi	19.79	49.12	1029.8	2023-06-06 23:44:36.418758	sent
3	pi	19.79	49.12	1029.77	2023-06-06 23:44:38.419972	sent
4	pi	19.78	49.15	1029.83	2023-06-06 23:44:40.415055	sent
5	pi	19.78	49.15	1029.88	2023-06-06 23:44:42.421841	sent
6	pi	19.77	49.16	1029.83	2023-06-06 23:44:44.414442	sent
7	pi	19.77	49.17	1029.8	2023-06-06 23:44:46.432448	sent
8	pi	19.76	49.17	1029.88	2023-06-06 23:44:48.415606	sent
9	pi	19.77	49.18	1029.85	2023-06-06 23:44:50.445589	sent
10	pi	19.71	49.25	1029.93	2023-06-06 23:45:22.414671	sent
11	pi	19.71	49.29	1029.93	2023-06-06 23:45:24.413073	sent
12	pi	19.79	49.13	1029.77	2023-06-06 23:49:30.412664	sent
13	pi	19.8	49.12	1029.72	2023-06-06 23:49:32.486401	sent
14	pi	19.8	49.12	1029.75	2023-06-06 23:49:34.561758	sent
15	pi	19.66	49.64	1029.99	2023-06-06 23:52:20.450558	sent
16	pi	19.65	49.81	1030.04	2023-06-06 23:52:22.432828	sent
17	pi	19.65	49.66	1029.99	2023-06-06 23:52:24.422289	sent
18	pi	19.65	49.58	1029.99	2023-06-06 23:52:50.495432	sent
19	pi	19.65	49.58	1030.07	2023-06-06 23:52:52.418053	sent
20	pi	19.65	49.56	1030.01	2023-06-06 23:52:54.415261	sent
21	pi	19.66	49.54	1030.07	2023-06-06 23:52:56.738010	sent
22	pi	19.64	49.51	1029.96	2023-06-06 23:52:58.438761	sent
23	pi	19.64	49.52	1030.04	2023-06-06 23:53:00.442353	sent
24	pi	19.65	49.52	1030.01	2023-06-06 23:53:02.448903	sent
25	pi	19.65	49.51	1030.01	2023-06-06 23:53:04.427306	sent

1 - 26 of 2287 Go to: 1



DB Browser for SQLite - /home/iot/minoriot-setup

Table: pending_data

	deviceID	temperature	humidity	pressure	rasptimestamp	status
314	pi	19.73	50.8	1029.8	2023-06-07 00:09:56.395671	sent
315	pi	19.73	50.77	1029.8	2023-06-07 00:09:58.390322	sent
316	pi	19.71	50.77	1029.74	2023-06-07 00:10:00.391511	sent
317	pi	19.72	50.7	1029.8	2023-06-07 00:10:02.394787	sent
318	pi	19.72	50.71	1029.88	2023-06-07 00:10:04.394447	sent
319	pi	19.71	50.74	1029.8	2023-06-07 00:10:06.389778	sent
320	pi	19.7	50.68	1029.82	2023-06-07 00:10:08.390337	sent
321	pi	19.7	50.63	1029.88	2023-06-07 00:10:10.396582	sent
322	pi	19.69	50.55	1029.85	2023-06-07 00:10:12.395902	sent
323	pi	19.69	50.53	1029.96	2023-06-07 00:10:14.396449	sent
324	pi	19.69	50.62	1029.8	2023-06-07 00:10:16.402090	sent
325	pi	19.69	50.61	1029.88	2023-06-07 00:10:18.389480	sent
326	pi	19.69	50.51	1029.91	2023-06-07 00:10:20.390713	sent
327	pi	19.67	50.52	1029.91	2023-06-07 00:10:22.391959	sent
328	pi	19.67	50.53	1029.93	2023-06-07 00:10:24.392128	sent
329	pi	19.66	50.51	1029.9	2023-06-07 00:10:26.389921	sent
330	pi	19.66	50.52	1029.96	2023-06-07 00:10:28.394229	sent
331	pi	19.65	50.54	1030.01	2023-06-07 00:10:30.392961	sent
332	pi	19.64	50.54	1029.93	2023-06-07 00:10:32.390363	sent
333	pi	19.64	50.51	1029.93	2023-06-07 00:10:34.390134	sent
334	pi	19.63	50.5	1029.93	2023-06-07 00:10:36.390309	sent
335	pi	19.62	50.49	1029.96	2023-06-07 00:10:38.392037	sent
336	pi	19.61	50.44	1029.96	2023-06-07 00:10:40.396578	Pending

313 - 336 of 336

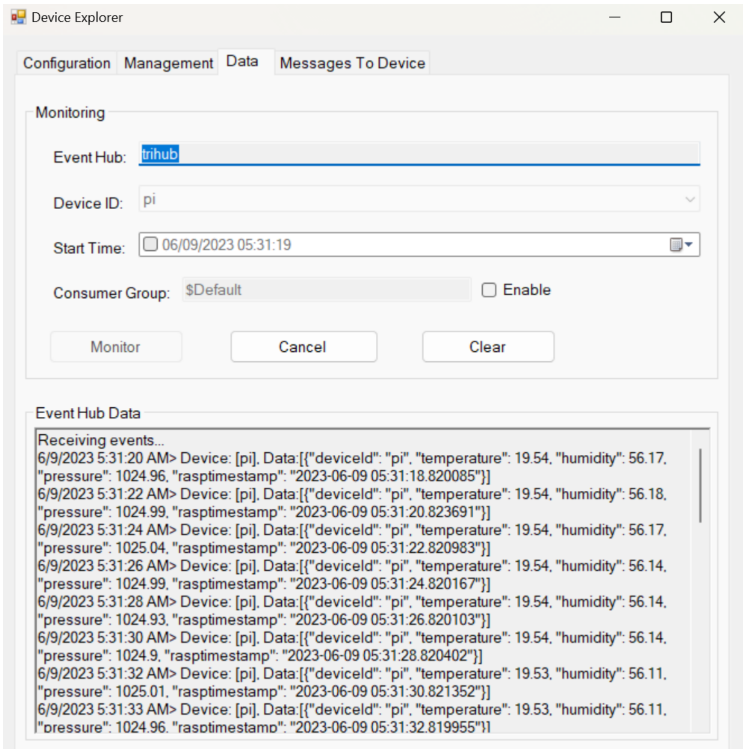
iothub.py - minoriot-setup - Visual Studio Code

```python
    if not ARGS.connection:  # If no connection string provided
        log.error("IOTHUB_DEVICE_CONNECTION_STRING in config.py variable or argument not found

    with console.status("Connecting to IoT Hub with Connection String", spinner="arc", spinner
        # Create instance of the device client using the connection string
        device_client = IoTHubDeviceClient.create_from_connection_string(ARGS.connection, conn

        try:
            # Connect the device client.
            device_client.connect()
        except Exception as e:
            log.error("Failed to connect to IoT Hub:", e)

        log.success("Connected to IoT Hub")


    try:
        while True:
            with console.status("Sending message to IoTHub...", spinner="bouncingBar"):
                # Send data to IoT Hub
                mqttc.on_message = store_locally
```

```
[+] Pending: deviceID=pi, temp=19.62, hum=50.49, pres=1029.96, time=2023-06-07 00:10:38.392037
[...] Sending message to Iot Hub
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 19.62,
    'humidity': 50.49,
    'pressure': 1029.96,
    'rasptimestamp': '2023-06-07 00:10:38.392037'
}
[+] Pending: deviceID=pi, temp=19.61, hum=50.44, pres=1029.96, time=2023-06-07 00:10:40.396578
  Waiting 2000ms  iot@T-pi:~/minoriot-setup $ cd /home/iot/minoriot-setup ; /usr/bin/env /bin/python /home/iot/.vsc
ode/extensions/ms-python.python-2023.4.1/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 39519 -- /home
/iot/minoriot-setup/iothub.py
[+] Connected to MQTT broker!
[+] Connected to IoT Hub
[...] Sending message to Iot Hub
[+] Message successfully sent!
{
    'deviceId': 'pi',
    'temperature': 19.61,
    'humidity': 50.44,
    'pressure': 1029.96,
    'rasptimestamp': '2023-06-07 00:10:40.396578'
}
[+] Pending: deviceID=pi, temp=19.61, hum=50.26, pres=1029.96, time=2023-06-07 00:14:18.404955
[...] Sending message to Iot Hub
[+] Pending: deviceID=pi, temp=19.61, hum=50.26, pres=1030.07, time=2023-06-07 00:14:20.384060
```

The image for the pending process described, the latest data sent to the database as pending, when it online again, it prioritized the pending data before the new data, new data then stored to the database as pending again, the loop kept going and going.

Use Iot hub explorer to check the data sent to iot hub:



Event Hub: trihub

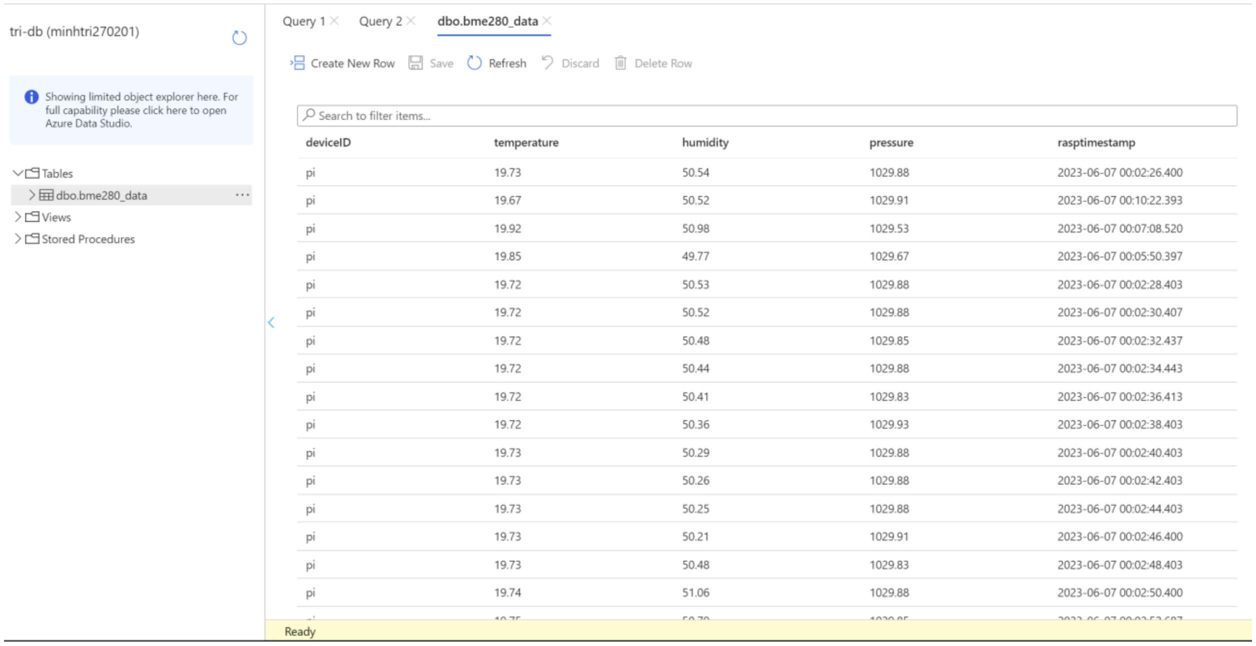
Device ID: pi

Start Time: 06/09/2023 05:31:19

Consumer Group: $Default   Enable

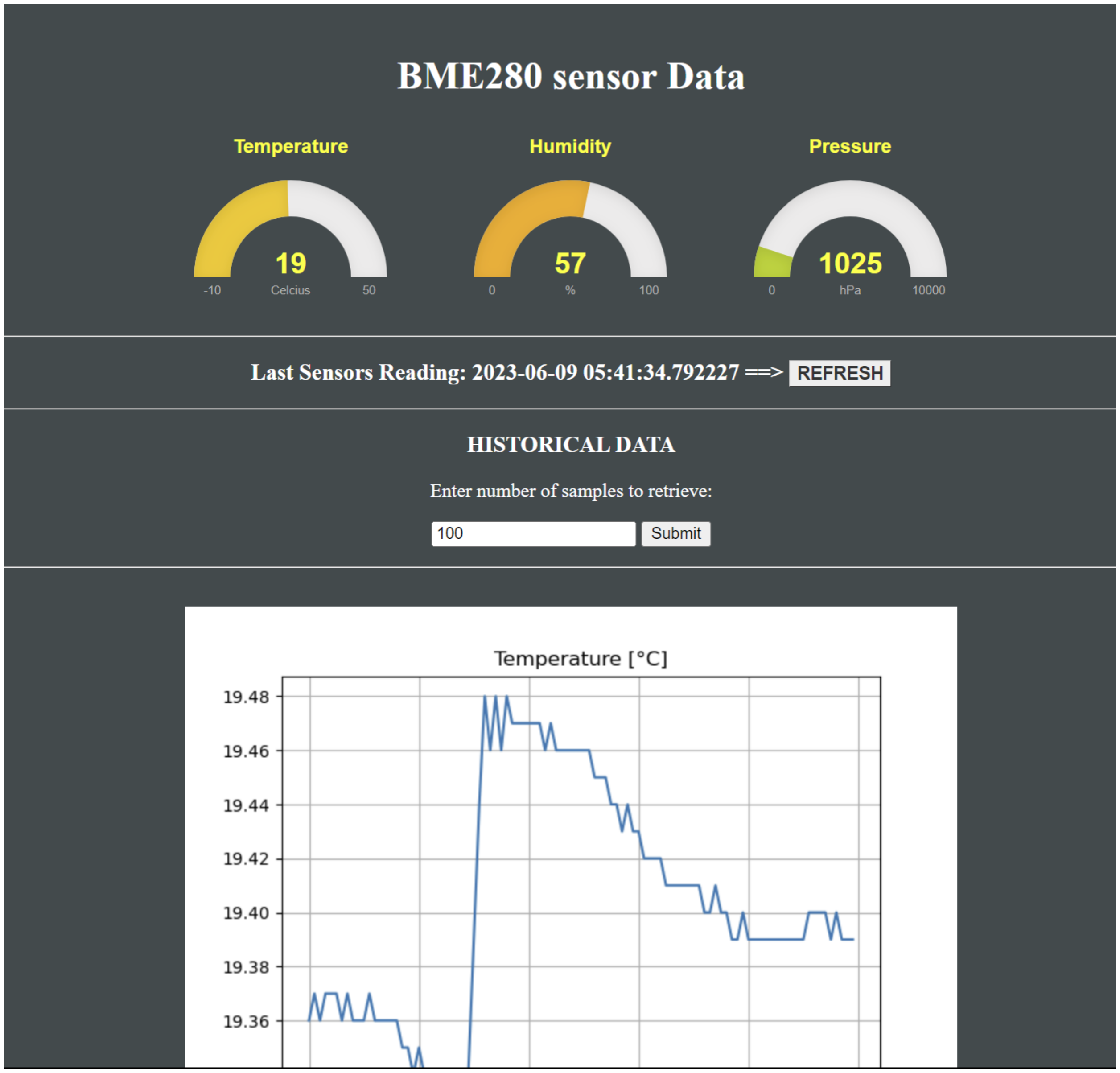
```
Receiving events...
6/9/2023 5:31:20 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.17,
"pressure": 1024.96, "rasptimestamp": "2023-06-09 05:31:18.820085"}]
6/9/2023 5:31:22 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.18,
"pressure": 1024.99, "rasptimestamp": "2023-06-09 05:31:20.823691"}]
6/9/2023 5:31:24 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.17,
"pressure": 1025.04, "rasptimestamp": "2023-06-09 05:31:22.820983"}]
6/9/2023 5:31:26 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.14,
"pressure": 1024.99, "rasptimestamp": "2023-06-09 05:31:24.820167"}]
6/9/2023 5:31:28 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.14,
"pressure": 1024.93, "rasptimestamp": "2023-06-09 05:31:26.820103"}]
6/9/2023 5:31:30 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.54, "humidity": 56.14,
"pressure": 1024.9, "rasptimestamp": "2023-06-09 05:31:28.820402"}]
6/9/2023 5:31:32 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.53, "humidity": 56.11,
"pressure": 1025.01, "rasptimestamp": "2023-06-09 05:31:30.821352"}]
6/9/2023 5:31:33 AM> Device: [pi], Data:[{"deviceId": "pi", "temperature": 19.53, "humidity": 56.11,
"pressure": 1024.96, "rasptimestamp": "2023-06-09 05:31:32.819955"}]
```

Start the stream analytics job on Azure to stream the data from the iothub (inputstream) to the database (outputstream). Open the database to check if the data is received and saved in the cloud database service.



deviceID	temperature	humidity	pressure	rasptimestamp
pi	19.73	50.54	1029.88	2023-06-07 00:02:26.400
pi	19.67	50.52	1029.91	2023-06-07 00:10:22.393
pi	19.92	50.98	1029.53	2023-06-07 00:07:08.520
pi	19.85	49.77	1029.67	2023-06-07 00:05:50.397
pi	19.72	50.53	1029.88	2023-06-07 00:02:28.403
pi	19.72	50.52	1029.88	2023-06-07 00:02:30.407
pi	19.72	50.48	1029.85	2023-06-07 00:02:32.437
pi	19.72	50.44	1029.88	2023-06-07 00:02:34.443
pi	19.72	50.41	1029.83	2023-06-07 00:02:36.413
pi	19.72	50.36	1029.93	2023-06-07 00:02:38.403
pi	19.73	50.29	1029.88	2023-06-07 00:02:40.403
pi	19.73	50.26	1029.88	2023-06-07 00:02:42.403
pi	19.73	50.25	1029.88	2023-06-07 00:02:44.403
pi	19.73	50.21	1029.91	2023-06-07 00:02:46.400
pi	19.73	50.48	1029.83	2023-06-07 00:02:48.403
pi	19.74	51.06	1029.88	2023-06-07 00:02:50.400

The program worked as expected. The data saved in Azure can also be exported as csv or json file for any kind of data analytic job. Now run the webserver.py to create the html link to the latest data. Login to the webserver as raspberry pi IP address:2704 (http://192.168.137.40:2704/)


BME280 sensor Data

Last Sensors Reading: 2023-06-09 05:41:34.792227 ==> REFRESH

HISTORICAL DATA

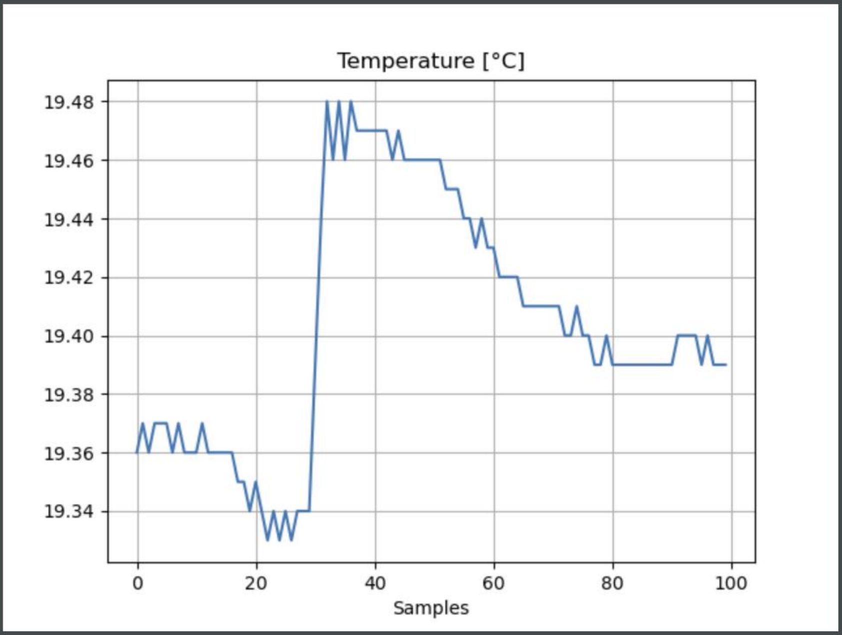
Enter number of samples to retrieve:

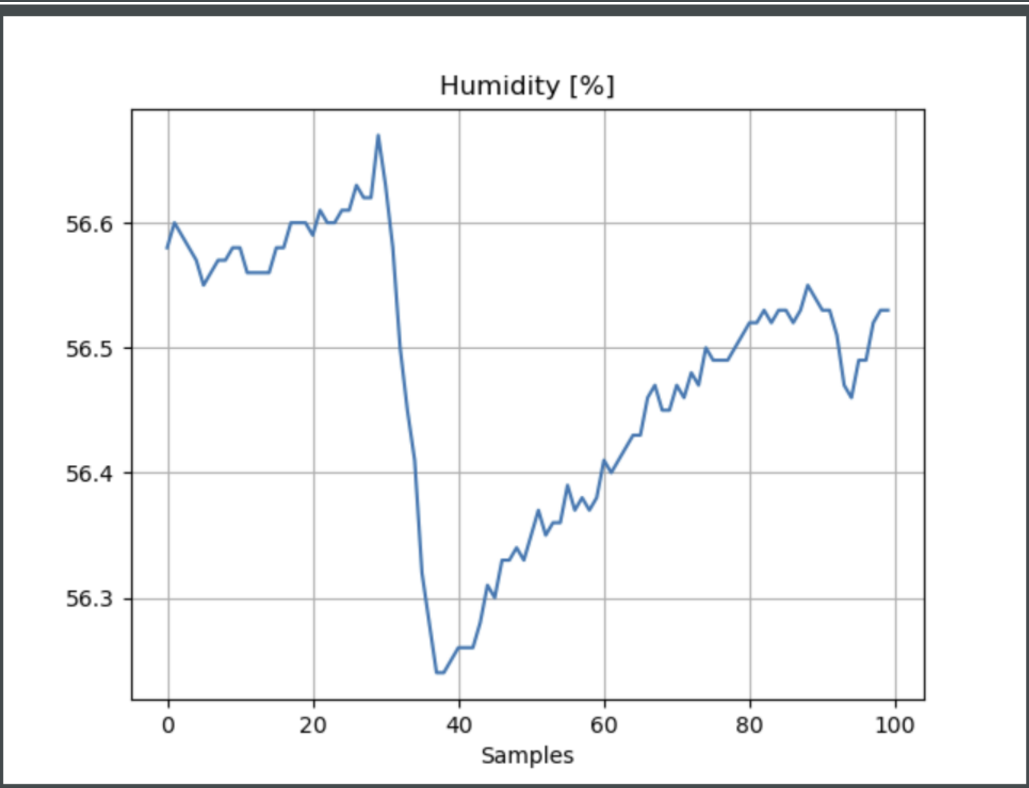
100 Submit

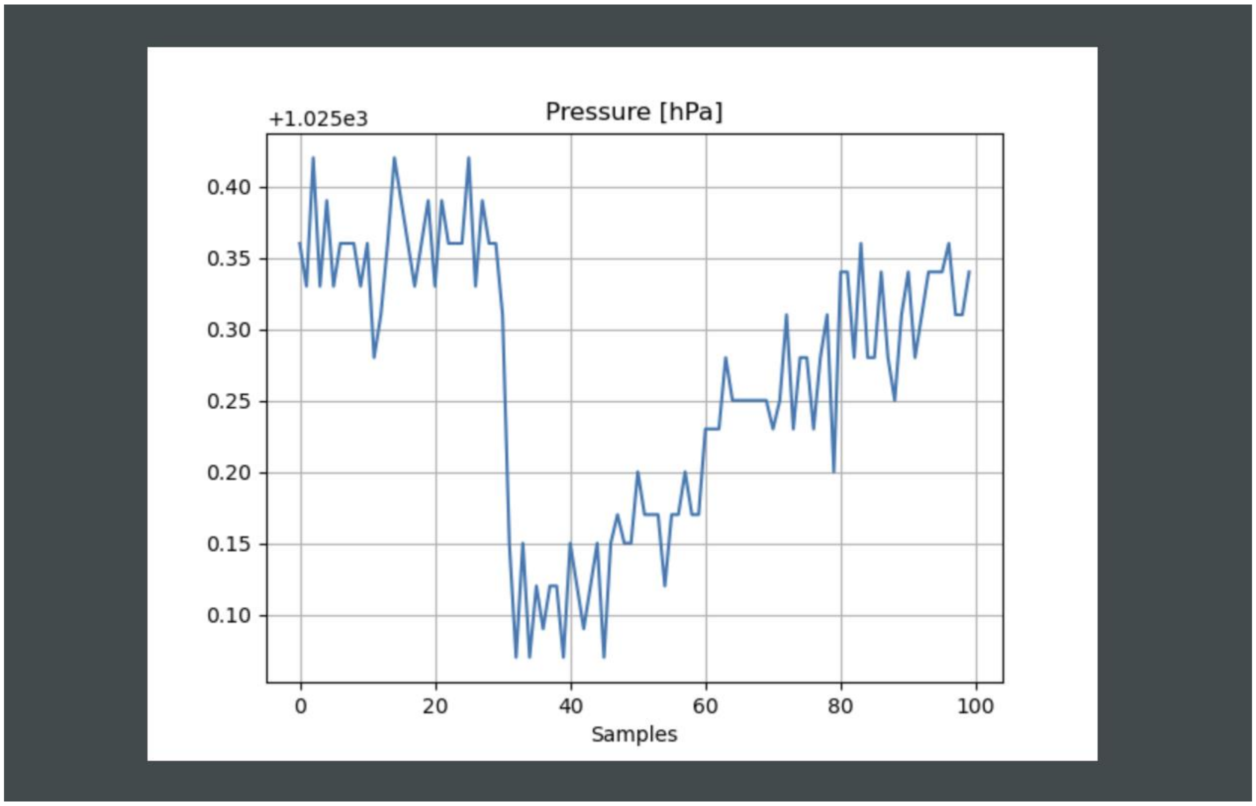
HISTORICAL DATA

Enter number of samples to retrieve:

100 Submit







For the gages, we can press refresh button to update the latest value. We can select the number of datapoint to plot and show the graphs with corresponding datapoint. The image above shows the graphs of 100 datapoints.

5 DISCUSSION

The program worked as expected.

No	Requirements	Acknowledge
1	BME280 – ESP32	ACK
2	ESP32 – RASPBERRY PI WIFI-NETWORK	ACK
3	SEND DATA TO BROKER USING MQTT	ACK
4	CREATE DATABASE AND STORE LOCALLY ON RASPBERRY PI	ACK
5	SEND TO AZURE IOT HUB	ACK
6	THE DATA LOSS PREVENTING PROCESS	ACK
7	WEBSERVER VISUALISE REAL-TIME DATA	ACK