Internet of Things

Contents

[2 INTRODUCTION 2](#_Toc137187209)

[3 DESIGN & IMPLEMENTATION 3](#_Toc137187210)

[3.1 SHOW AN OVERVIEW IN A DIAGRAM OF THE COMPLETE ARCHITECTURE OF YOUR IOT SYSTEM. 3](#_Toc137187211)

[3.2 CONNECT SENSOR BME280 TO THE ESP32. COLLECT TEMPERATURE, HUMIDITY, AND PRESSURE. 3](#_Toc137187212)

[3.3 SECURITY ESP32. 4](#_Toc137187213)

[3.4 CONNECT THE ESP32 TO THE RASPBERRY PI BY A WIFI-NETWORK AND SEND THE DATA TO THE BROKER USING MQTT PROTOCOL. 5](#_Toc137187214)

[3.5 SECURE THE MQTT SERVER AND THE DATA STREAM TO THE SERVER. 9](#_Toc137187215)

[3.6 CREATE A DATABASE ON THE RASPBERRY PI TO STORE DATA LOCALLY. 9](#_Toc137187216)

[3.7 CONNECT THE RASPBERRY PI TO THE IOT HUB. 10](#_Toc137187217)

[3.8 WEBSERVER 12](#_Toc137187218)

[4 Result 18](#_Toc137187219)

[5 DISCUSSION 25](#_Toc137187220)

# 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 |  |

# DESIGN & IMPLEMENTATION

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

A screenshot of a computer

Description automatically generated with medium confidence

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.

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

*Flowchart:*

Diagram

Description automatically generated

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:*

Text

Description automatically generated

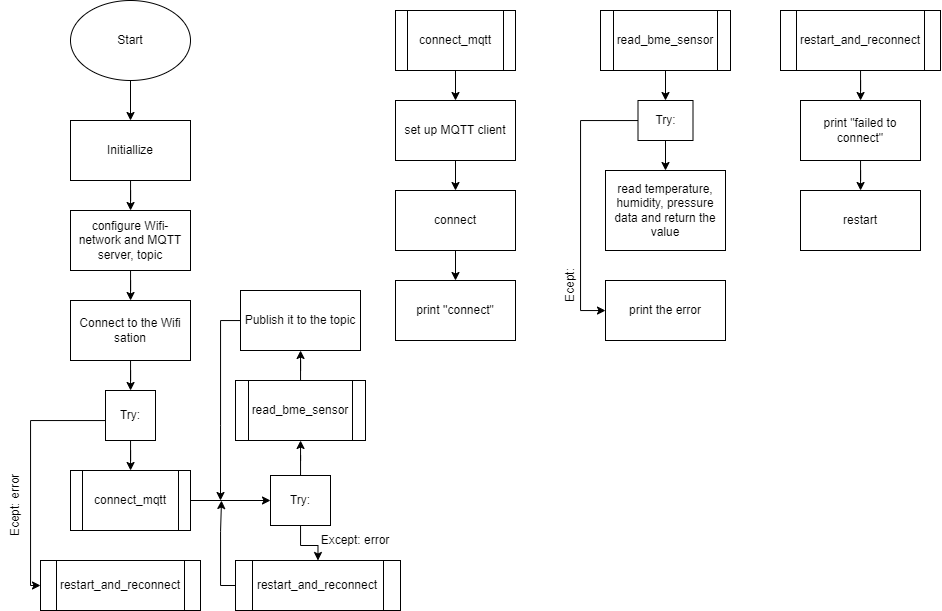
## SECURITY ESP32.

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.

## 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:*

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

*Raspberry Pi flowchart:*

A picture containing text, diagram, plan, technical drawing

Description automatically generated

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:*

A screenshot of a computer program

Description automatically generated with medium confidence

A screen shot of a computer program

Description automatically generated with low confidence

## 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>

## 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:

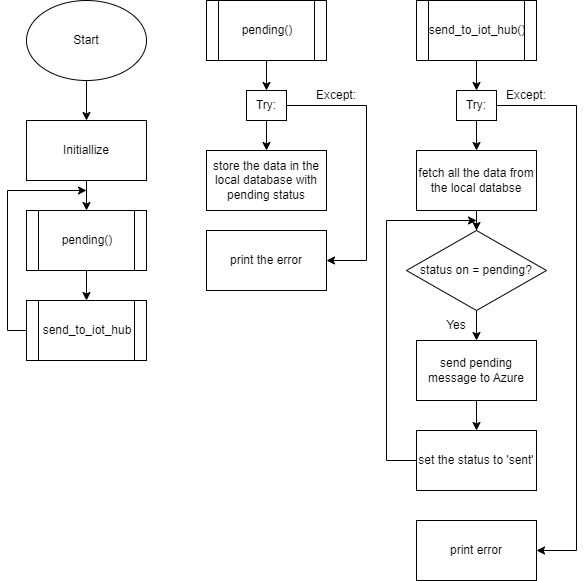
BEGIN;

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.

## 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:*

A picture containing text, screenshot, font

Description automatically generated

A screen shot of a computer program

Description automatically generated with low confidence

A picture containing text, screenshot

Description automatically generated

## 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*

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with medium confidence

*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

}

# Result

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

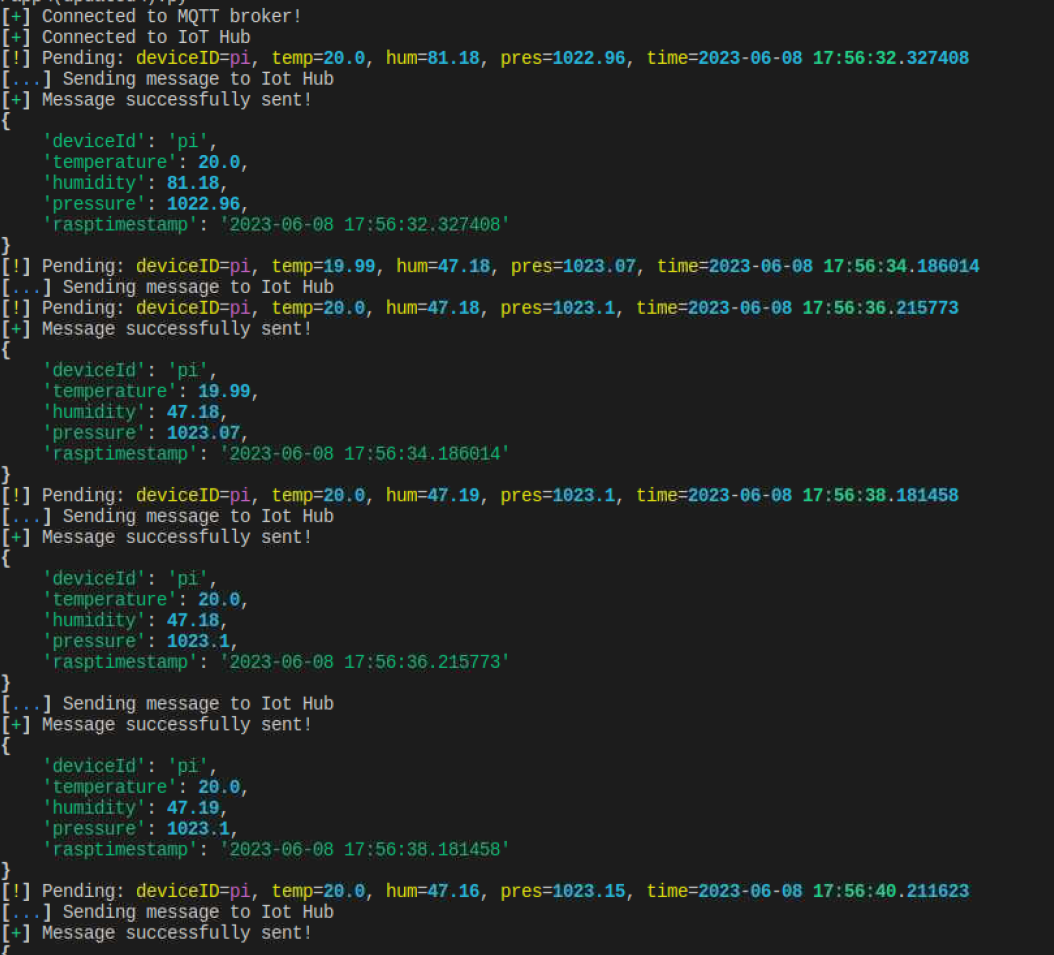
A screen shot of a computer

Description automatically generated with medium confidenceUse MQTT explorer to check the data sent to the broker.

A screenshot of a computer

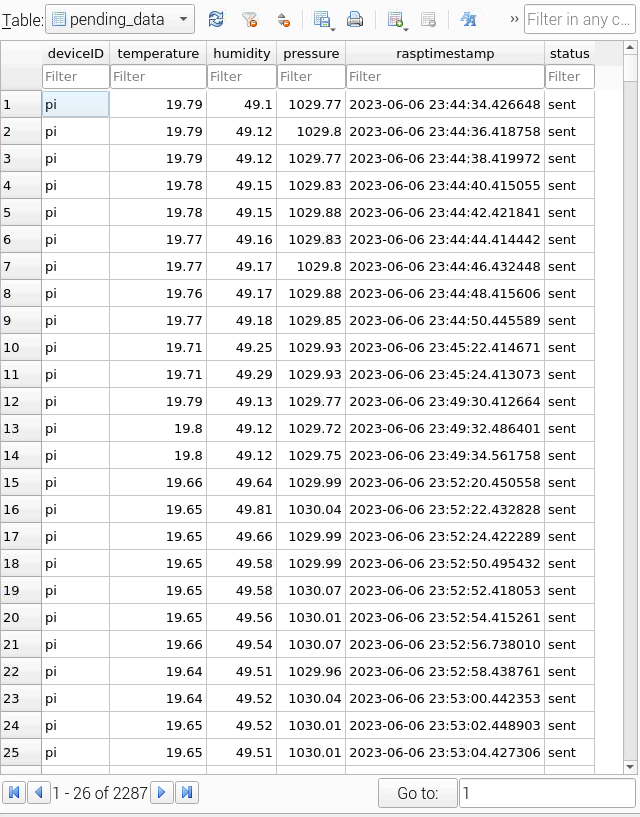
Description automatically generated

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



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:





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:

A screenshot of a device

Description automatically generated with medium confidence

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.

A screenshot of a computer

Description automatically generated with medium confidence

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/>)

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a graph

Description automatically generated with low confidenceA picture containing line, plot, text, diagram

Description automatically generatedA picture containing text, plot, line, number

Description automatically generated

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.

# 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 |