

Bachelor's thesis

Information and Communication Technology

2023

Vinayak Chaturvedi

# Data Communication using MQTT with BME688 sensor



Bachelor's Thesis | Abstract

Turku University of Applied Sciences

Information and Communication Technology

2023 | 57

Vinayak Chaturvedi

## DATA COMMUNICATION USING MQTT WITH BME688 SENSOR

In today's era of technology, there are an abundant variety of integrated sensors which range in a variety of low-cost sensors, which has encouraged the demand for more advanced sensor performance. This thesis describes the sensor in a much more detailed view which is the BME688 - which is capable of sensing gas, pressure, humidity, and air quality. Following to it, a project based description on BME688 which includes various hardware and software components which highlights the use of the sensor.

The implementation of the project started from the hardware side comprising of building the circuit, by connecting the ESP32 microcontroller with the BME688 sensor with male-female jump wires. Simultaneously, working on the software side by integrating Raspberry Pi with the computer to download MQTT Mosquitto Broker and Node-RED dashboard, through PuTTY software to keep the files and data encrypted.

After the completion of the thesis project, all the published data were adequate, the implementation done on the hardware side was also working efficiently, and getting the right results in the correct range was also been observed. All the hardware components were showing the best abilities to communicate with each other. Discussing the software side, that was also very compatible and so the data communication throughout the project was implemented easily.

Keywords:

BME688 Sensor, ESP32, Raspberry Pi, MQTT, Mosquitto Broker, puTTY.

# Content

<b>List of abbreviations</b>	<b>6</b>
<b>1 Introduction</b>	<b>7</b>
<b>2 Hardware and Software side of the project</b>	<b>9</b>
2.1 ESP32	9
2.1.1 ESP-32 Pin Layout	11
2.2 Raspberry Pi	13
2.2.1 Applications of Raspberry Pi	14
2.2.2 Pros and Cons of RaspberryPi	15
2.2.3 Raspberry Pi board Layout	16
2.3 Sparkfun BME688 Sensor	19
2.3.1 Sparkfun BME688 Sensor Pin Layout	21
2.4 Bread board and Jumpwires	23
2.5 MQTT	25
2.5.1 Message Architecture	25
2.5.2 MQTT Architecture	26
2.5.3 MQTT Broker	27
2.5.4 Mosquitto Broker	28
2.6 Node-Red	28
2.7 PuTTY Software	29
<b>3 Implementation</b>	<b>31</b>
3.1 Circuit Design of the Project	32
3.2 Working details on software side	33
<b>4 RESULT</b>	<b>51</b>
<b>5 CONCLUSION</b>	<b>54</b>
<b>References</b>	<b>55</b>

## Figures

Figure 1 ESP32	11
Figure 2 QFN(Quad Flat No Leads)	11
Figure 3 ESP32 PinOut	13
Figure 4 Raspberry Pi	14
Figure 5 Raspberry Pi Pinout	16
Figure 6 Raspberry Pi - 4	17
Figure 7 Processor	17
Figure 8 GPIO Pins	19
Figure 9 Sparkfun BME688 Sensor	20
Figure 10 Power Supply	21
Figure 11 Qwiic and I2C	21
Figure 12 SPI	22
Figure 13 Power LED Jumper	22
Figure 14 CSB Jumper	23
Figure 15 Breadboard	24
Figure 16 Male - Male Jumpwires	24
Figure 17 Female - Female Jumpwires	25
Figure 18 MQTT Architecture	27
Figure 19 Mosquitto Broker	28
Figure 20 Node-RED Working	29
Figure 21 Thesis Project Idea	31
Figure 22 Working Circuit	32
Figure 23 Clicking add on library to install ESP32	33
Figure 24 Inserting a URL for downloading the ESP32	33
Figure 25 Selecting board from Board Manager	34
Figure 26 Installing ESP32 by Espressif Systems	34
Figure 27 ESP32 Installed	35
Figure 28 Selecting board for testing the installation of ESP32	35
Figure 29 Selecting COM4 port	36
Figure 30 Selecting WiFi Modules	36

Figure 31 New Sketch windows of WiFiScan	37
Figure 32 Done Uploading	38
Figure 33 Successfull working of ESP32 in COM4 Port	39
Figure 34 PuTTY software download	39
Figure 35 Selecting hostname and connection type in PuTTY software	40
Figure 36 Logging in as pi and password	41
Figure 37 Opening of the Terminal window of PuTTY	41
Figure 38 Mosquitto Broker version command	42
Figure 39 Hostname to get the Raspberry Pi IP address	43
Figure 40 Installation of necessary modules of Node-RED	44
Figure 41 Implementations of Sequence for Node-RED	45
Figure 42 Node-RED Start	46
Figure 43 IP Address	47
Figure 44 Node-RED login page	47
Figure 45 Connection of Raspberry Pi and MQTT broker	48
Figure 46 Getting BME688 readings	48
Figure 47 Connection of the WiFi	49
Figure 48 Getting readings from BME688 after regular interval	50
Figure 49 Serial monitor result published	51
Figure 50 Node-RED Connection	52
Figure 51 Circuit implementation	53

## List of abbreviations

Abbreviation	Explanation of abbreviation (Source)
BME688	BOSCH manufactured AI Gas sensor.
ESP32	Microcontroller
MQTT	Message Queuing Telemetry Transport
Raspberry Pi	Single Board Computers
Mosquitto Broker	Open source message broker in MQTT protocol
PuTTY	Free open source terminal emulator
Node-RED	Programming tool for connecting nodes virtually

# 1 Introduction

In today's rapid growth of technologies, various kinds of sensors have been built and implemented to fetch data. As time has passed the focus has been more on the accuracy of sensing the data. These sensors could be found everywhere, the whole world is full of sensors and their applications. Ranging from your mobile phones to smart homes all the devices are sensors, which transmit the data at a defined interval of time. Sensors could be categorized by various characteristics such as applications, the material used, energy consumption, cost, etc.

The main idea of this thesis is to communicate data using MQTT with BME688 sensor, which is been connected to the ESP32 microcontroller followed by Raspberry Pi working on the backend side of the project. The project would be following an MQTT protocol built in a Pub - Sub model with an established MQTT broker which creates a bridge between the transmitter and receiver while the sensor is in its working stage. The final data collected would be displayed in a Node-Red dashboard, programmed in Arduino IDE. Specifically choosing BME688 Gas sensor for this project as it is capable for sensing Gas, Humidity and Pressure, which gives more options in it's applications. All the terms and concepts would be explained in more detail in further this review.

At the end, the thesis literature review is been divided into 5 chapters each explaining the concepts used in this project in a detailed information.

The first chapter gives an introduction about the thesis idea and following to that a thesis overview. Having an aim of illustrating all the complexities in a understandable manner.

Second chapter gives a light on the hardware and software side of the project which was used during the project. Giving all the technical knowledge and expertise by explaining it more with pin layout diagram and screenshots of downloading and using the software for the other side of the project.

Third chapter is the implementation where all the approach and ideas are been discussed, and then executed.

Fourth chapter shares the results occurred while testing it throughout the process.

Fifth chapter concludes the thesis.



## 2 Hardware and Software side of the project

This project comprises of two sections, hardware and software. In the hardware side of the project many components are been used such as ESP-32, Raspberry Pi, BME688 gas sensor, breadboard, jumpwires, ethernet cable, etc. On the other section in the software side many softwares, dashboards, IDE were been used. All the mentioned technologies above will be explained in detailed view in the upcoming chapters.

### 2.1 ESP32

ESP-32 which is created by Espressif Systems is an SoC microcontroller loaded with various modules which are Low-cost and Low in power consumption. ESP-32 is the next generation of ESP8266 which is quite more advanced and with much more features, as the ESP-32 has in-built Wi-Fi, Bluetooth, and Bluetooth Low Energy(BLE). ESP-32 uses a Tensilica Xtensa LX6 dual microprocessor which can operate with a frequency of up to 240mHZ. ESP-32 microprocessor has a lot of in-built packages with a high level of integrations such as-

- Antenna Switches
- Power Amplifier
- Low noise reception amplifier.[1]

above all the mentioned information and features, ESP-32's current usage is less than 5 $\mu$ A.

Some important specs of the latest ESP-32 variant which make it a more essential hardware component used in many projects are-

- Integrated SPI Flash - 4MB
- ROM - 448KB (For booting and core functions)
- Operating Voltage- 3.3V

- Operating Current - 80mA(which is the average current)
- Operating temperature ranges - from - 40 degrees to 85 degrees
- On-chip-sensor-Hall Sensor

Due to its amazing built in features of Wi-Fi and Bluetooth, there is no need to install external/separate packages which makes it a game changer in the market. ESP-32 allows to access the Wi-Fi Access Point and the Station mode. Its compatibility with the Internet Protocol (IP) such as - HTTP, MQTT, and TCP/IP also makes it more advanced. [1]

For security purposes too, ESP-32 also supports HTTPS which makes the transmission of data processes more secure and well-encrypted by both end devices. The support is provided for various applications such as LTE, LoRa Modules, etc.

Successful hardware like ESP-32 is also quite user-friendly as its programmed codes can be run in various IDE such as ArduinoIDE, PlatformIO, Micropython, Espressif IDF( IoT Development Framework), etc.

A Majority of the programmers use ArduinoIDE as it's more efficient and installing or importing packages from external sources is also quite easy. [1]

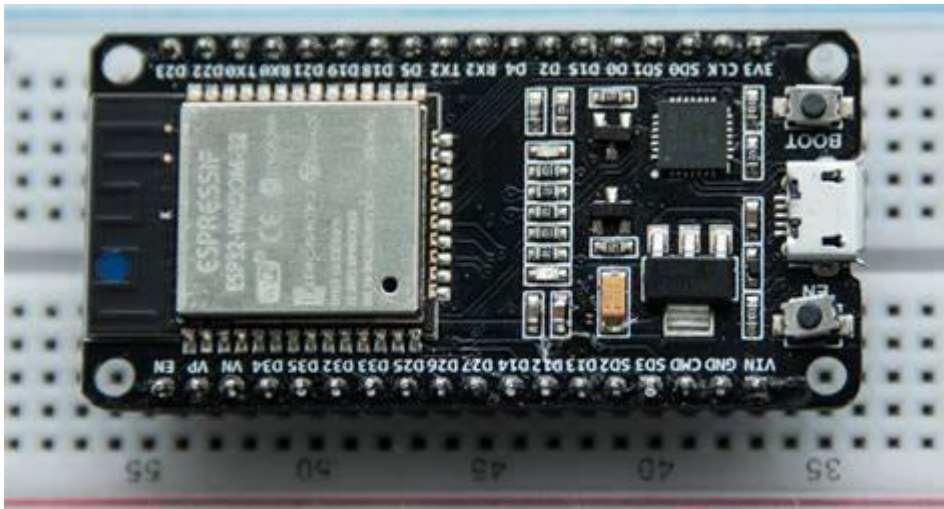


Figure 1 ESP32 [1]

### 2.1.1 ESP-32 Pin Layout

The ESP-32 microcontroller IC is available with 48 pins QFN(Quad Flat No Leads) in Figure – 2, package it is quite difficult to solder the IC on the PCB board. ESP-WROOM-32 also contains a 4MB SPI Flash IC, a 40Mhz of Oscillator, a PCB Antenna, and some other components for the microprocessor to work. [3]



Figure 2 QFN(Quad Flat No Leads)

ESP-32 Peripheral components:-

- 34 Programmable GPIOs

- 16 RTC GPIOs
- 18 12-Bit ADC channels
- 2 8-Bit DAC channels
- 3 UART Interfaces
- 3 SPI Interfaces
- 2 I2C Interfaces
- 2 I2S Interfaces
- 10 Capacitive Touch Sensing GPIOs
- 16 PWM Channels

GPIO - The most common and important peripheral is the GPIO. In ESP-32, 34 GPIO pins carry more than one function. [3]

SPI - The ESP-32 Wi-Fi has 3 slots including (SPI, HSPI, and VSPI). The SPI is used for Flash Memory so we have two interfaces.

I2C - There are 2 I2C pins in the ESP-32 which are assigned in pins SCL and SDA. If using Arduino IDE then the default pins are-

SDA - GPIO21

SCL - GPIO22

Interruption - All the pins in ESP-32 are very much capable of interruption.[3]

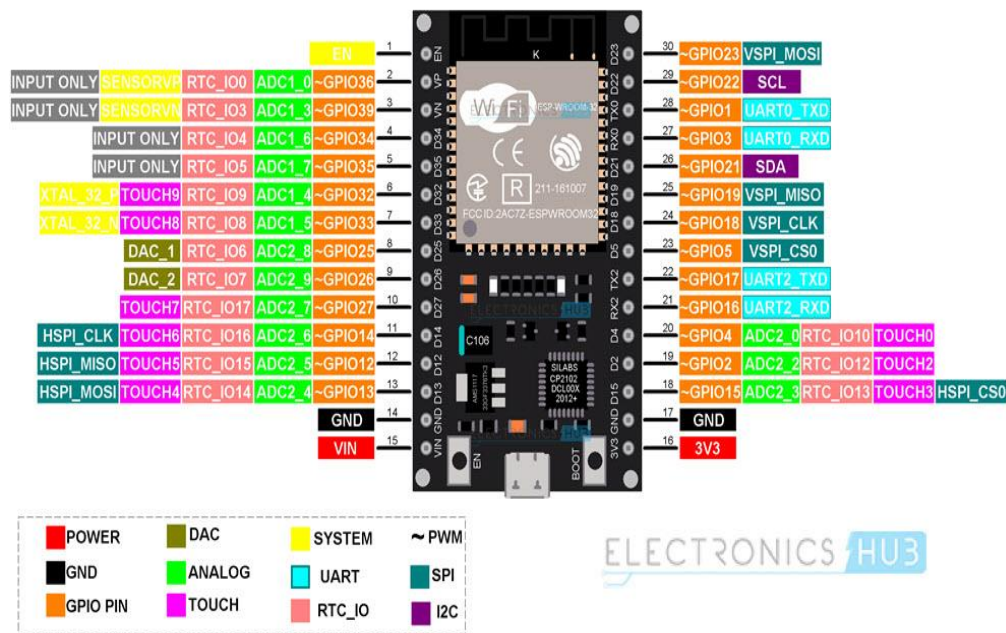


Figure 3 ESP32 PinOut [2]

## 2.2 Raspberry Pi

A microprocessor which was founded in the year 2008 used to develop a solution with low-cost maintenance, a compact credit-size card computer that inspired many young coders to code and access. It's quite compatible with other electronic devices such as it connects easily with computer monitors, and TVs using a standard keyboard and mouse.

Working with raspberry pi has many advantages, as it can be programmed by various programming languages like - Python, and Scratch. This small credit size computer can do everything and anything which we expect from a Desktop computer. We can browse the internet, make spreadsheets, and much more. It has skills to communicate with the outside environment and has been used like an array of digital devices. [4]

Raspberry Pi was initially built for younger generations so that they could program and can get familiar with the hardware component. Working with Raspberry Pi, we can make many projects such as - home automation, building

strong programming skills, use for industrial applications, and much more. Currently, in the market there are quite a few models of Raspberry Pi which can be operated in an open - source environment which is the Raspbian.

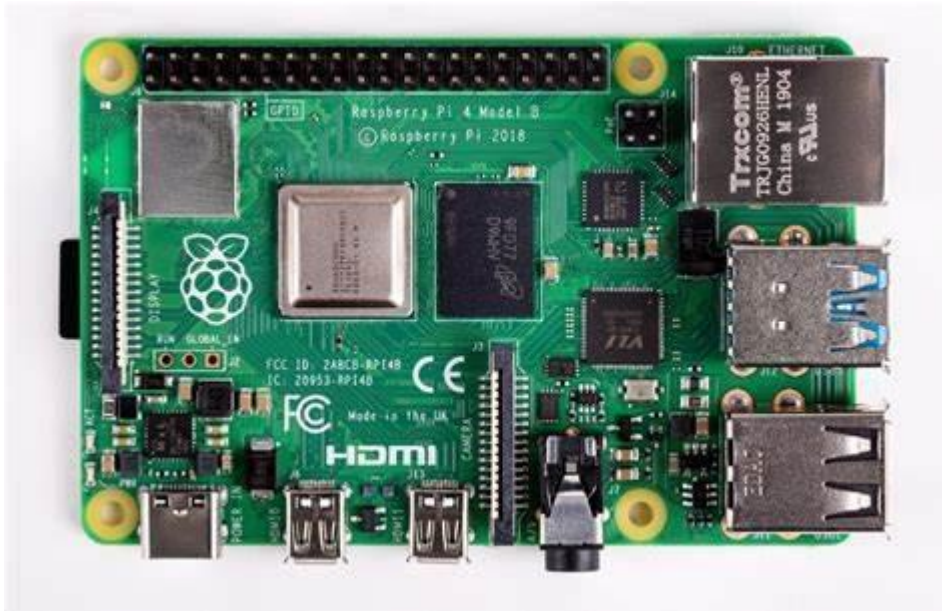


Figure 4 Raspberry Pi

### 2.2.1 Applications of Raspberry Pi

Raspberry Pi's main motive was to amuse people with computing, electronics, and programming. This microcontroller was so handy to use and it's fully loaded component made advanced projects have easy approaches. Some mentioned applications which are -

- Home Automation - The microcontroller is so compatible with other devices such as relay sensors and light, smartphones, and computers. The operator could access the system remotely.
- AI Assistant - It enables the language through its libraries from Google Assistant SDK, as well as Google cloud speech.
- Live Bots - Live bot is a system that enables the user to handle the bots based on Pi over the internet. [4]

## 2.2.2 Pros and Cons of RaspberryPi –

### Pros -

#### Multiple Sensors:

- Due to its multiple sensor compatibility, it can connect and support various sensors at a time, which motivates many electronics enthusiasts.
- Supports all kinds of Codes - This board supports all kinds of codes of any language which makes it more widely used in various fields of technology whether it's Python, C, C++ or Java, etc.
- It can also be used as a portable computer- If the user attaches a display/monitor to the Raspberry Pi it can become a portable computer and using various apps, sites and operating systems becomes possible.
- Fast Processor - Pi has a fast processor as compared to Arduino and other microcontroller modules. Pi's 1.6GHz processor in its 4B generation, on the other hand, Arduino is integrated into a controller which makes Pi a much more faster and effective option.
- Pi is so inexpensive and widely available that it can be used by many developers around the world. [5]

### Cons -

There are some demerits of the microcontroller while using it as follow-

- RTC - The board does not have an RTC (Real Time Clock) with a backup battery.
- Overheating - The board is been missed out on some heat conduction or fan, with such a pwerfull processor and multiple features the board starts heating up with an average usage of 6 - 7hrs. Compact size and missing heating dispersion cause the board the heated up.
- Not able to install Windows OS - The majority of the users use Windows as an operating system because windows are so user-friendly, for video editing and much more like software. Due to this reason, Pi has to compete with many counter devices.
- Missing Internal Storage - The major drawback of Raspberry Pi is that it does not have internal storage for the device. Since uinserting, a SD card acts as an internal storage for the device. [5]

### 2.2.3 Raspberry Pi board Layout –

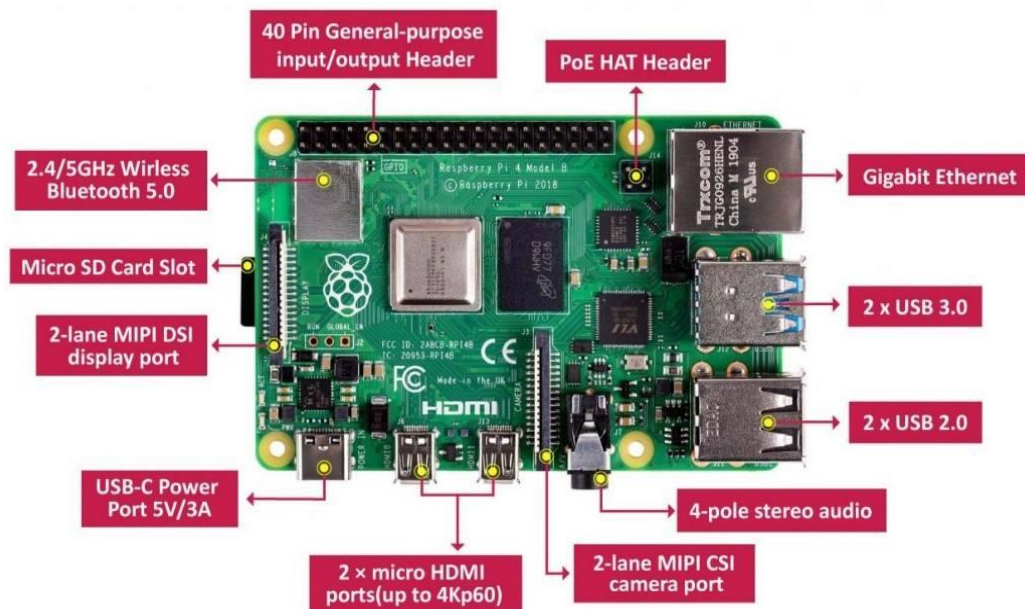


Figure 5 Raspberry Pi Pinout [6]

Raspberry Pi 4 Model B is the latest version of the Raspberry Pi which is much more advanced and with more advanced features. It was manufactured in the year 2019. Raspberry Pi runs a Debian - based operating system which is referred to as Raspbian, it is provided by the Raspberry Pi foundation.

Raspberry Pi 4 specifications :

USB type -C power input port.

It has a standard 40 GPIO headers.

It has a 64-bit quad-core processor having cortex-A72 clocked at 1.5GHz.

It has a dual-band 2.4/5.0GHz, Wi-Fi, Bluetooth 5.0, 2 USB 3.0, and 2 USB 2.0 ports. [6]





Figure 6 Raspberry Pi - 4

The Processor of RaspberryPi:

Model 4 has a much more powerful quad-core Cortex-A72, 64Bit CPU running at a frequency of 1.5GHz, and a broadband VideoCore VI GPU running at 0.5GHz. It can also play 4K videos and 2HDMI outputs. Adding, a fully functional Gigabit Ethernet interface and USB 3.0 port. [6]



Figure 7 Processor [5]

### Power Circuitry:

The Pi uses MXL7704 which was developed specifically for Raspberry Pi. The chip reduces the complexity of the pi board and also produces more power to all the areas to run with more power. In this model many components were reduced, so the cost of this model also decreased. [6]



Figure 8 Power Circuitry

There are 40 GPIO pins in raspberry pi 4. Which includes all types of pins like PWM, MISO - MOSI, 3V, and 5V power pin, TXD - RXD, and many other pins.

The board also has a separate connector for the camera module and display module where you can attach a display of the camera to the pi board. It also has a headphone jack for streaming audio. [6]

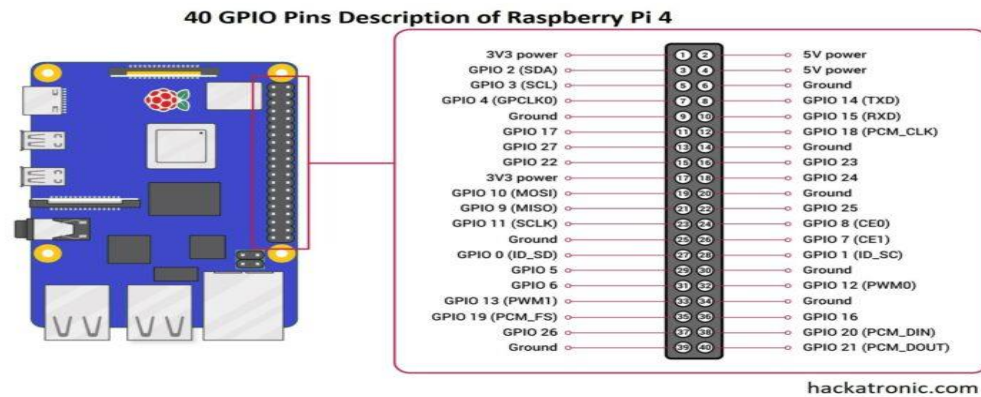


Figure 8 GPIO Pins [6]

### 2.3 Sparkfun BME688 Sensor

The BME688 Sensor is a breakout that senses temperature, gas, humidity, and barometric pressure. The gas sensor on the BME688 can detect a variety of Volatile Organic Compounds (VOCs), Volatile Sulfur Compounds (VSC), and other gases such as Carbon Monoxide, this sensor calculates in Parts Per Billion (ppb) range. This sensor communicates via I2C or SPI, both I2C and SPI pins are also broken out to standard 0.1 in spaced pins. Combining it with precise temperature, humidity, and barometric pressure and the BME688 can work as a completely standalone environmental sensor all in a 1 in. x1in. breakout. [7]

Some important features of the BME688 Sensor: -

Operating voltage range:

- 1.7V - 3.6V
- Typically 3.3V if using the Qwiic cable. [9]

Relative humidity:

- Operating range 0% to 100%

- Resolution: +- 0.008% RH
- Temperature
- Operating range: -40C to +85C
- Absolute Accuracy: +-1.0C
- Resolution: 0.01C [9]

Pressure :

- Operating range: 300hPa - 1100hPa
- Absolute Accuracy: +-60Pa (0C to 65C)
- Resolution: 0.01C

Gas :

- F1 score for H2S scanning: 0.92
- Standard scan speed: 10.8 sec/scan
- Sensor-to-sensor deviation: +-15% +- 15

Current consumption:

- 2.1  $\mu$ A to 18mA
- 0.15  $\mu$ A (sleep mode)
- 3.9 mA in standard gas scan mode. [9]



Figure 9 Sparkfun BME688 Sensor

### 2.3.1 Sparkfun BME688 Sensor Pin Layout

Power:

The BME688 sensor supplies a voltage between 1.71 to 3.6V. The power can be supplied to the board either through dedicated GND pins which are located on either side or in either one of the Qwiic connectors. [8]

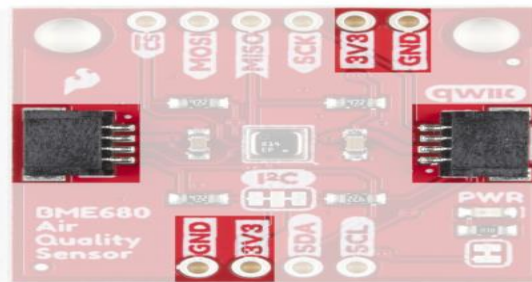


Figure 10 Power Supply

Qwiic and I2C Interface:

The Environmental sensor BME688 Qwiic communicates over I2C by default. It has routed the BME688's I2C pins to two Qwiic connectors as well as broken them out to 0.1. [8]

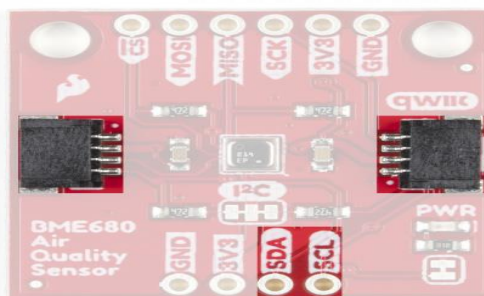


Figure 11 Qwiic and I2C

## Serial Peripheral Interface (SPI)

Communicating over SPI requires more connections than I2C but it is more versatile and can be much faster. To communicate with the BME688 Qwiic board over SPI, users will need to cut the CSB and ADR jump wires. [8]

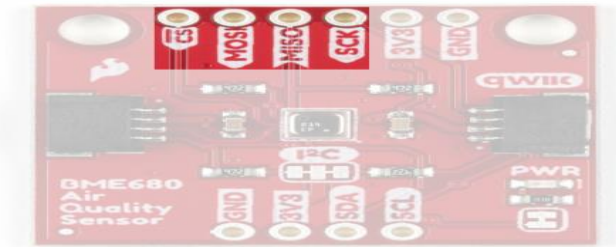


Figure 12 SPI

## Solder Jumpers

The BME688 sensor has 4 solder jumpers which can be modified to alter the functionality of the sensor. [8]

### Power LED Jumper

The jumper connects the power LED to 3.3V via a 1k ohm resistor. This jumper is usually closed so to disable the LED, sever the trace between the two pads. This is quite helpful to reduce the total current draw of your breakout for low-power applications. [8]

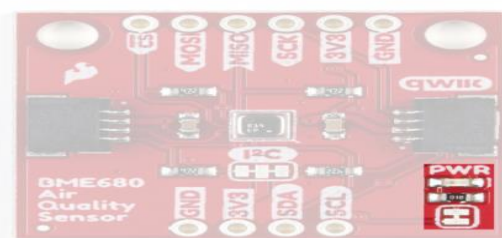


Figure 13 Power LED Jumper

## CSB Jumper

This jumper applies only to the BME688 Qwiic board. The CSB pin is pulled up to VDDIO to configure the board for I2C communications by default. To communicate with the sensor over SPI, the CSB jumper must be cut along with the ADR jumper [8]. Once the CSB pin has been pulled low during SPI communication, the sensor will communicate over SPI until there is a power



reset.

Figure 14 CSB Jumper [8]

## 2.4 Bread board and Jumpwires

A breadboard is a widely used tool for test and designing a circuit, with no need of soldering the components required. With working with the breadboard it is quite easy to dismantle all the components and reuse them. It consists of an array of conductive metal chips encased in a box of white plastic. A typical breadboard layout consists of two types of the region which are called Strips. Bus strips and Socket strips. The bus strip usually provides a power supply to the circuit. It has two columns, one for ground and the other for voltage. Socket strips are used to hold most of the components in a circuit, it consists of two sections each with 5 rows and 64 columns. Every column is connected electrically from the inside. [10]

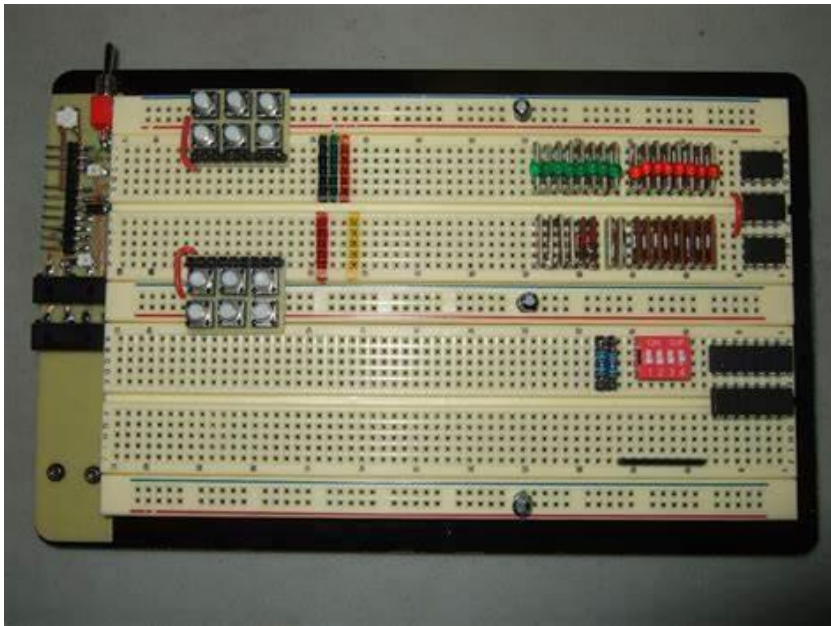


Figure 15 Breadboard

Jumpwires are simply wires that have a connector at each end, which allows them to connect to two points without soldering. These jump wires are used with breadboards and for other prototype tools, to change the circuit. These jump wires come in various colors but they don't mean anything all the wires are the same. There are two types of jump wires Male-Male and Female to Female wires. Male wires have pins at the end and are been used to plug with other components, while Female end does not and are used to plug into. [11]



Figure 16 Male - Male Jumpwires [12]





Figure 17 Female - Female Jumpwires [12]

## 2.5 MQTT

MQTT stands for Message Queuing Telemetry Transport. It is a lightweight messaging protocol for cases where clients need a small code and are connected to unreliable networks for networks with limited bandwidth, it is usually used with Machine-to-Machine communications and is often also used with many IoT-based projects. [13]

MQTT runs on a Pub/Sub model using TCP/IP topology. In this protocol, there are two types of systems: clients and brokers. Which will be discussed in the following chapters. MQTT is an event-driven protocol, there is no ongoing data transmission, and this keeps transmission to a minimum. A client can only publish data when there is information to be sent, and a broker only sends out information to the subscriber when new data arrives. [13]

### 2.5.1 Message Architecture

MQTT minimizes its transmission with a tightly defined small message construction. Every message has a fixed 2 bytes of header, and an optional header may be used but it increases the size of the message. There are three types of Quality of Service(QoS) levels that allow network designers to choose between minimizing data transmission and maximizing reliability.

QoS 0 - Offers a minimum amount of data transmission. With this level, each message is delivered to a subscriber once with no confirmation. In this way, it is not confirmed if the subscriber has received the message or not. This kind of method is referred to as a "Fire and Forget" message or "at most delivery". This is assumed because that delivery is complete, these messages are not stored for delivery to disconnected clients that later reconnect. [13]

QoS 1 - In this level of transmission the broker attempts to deliver the message and then waits for a confirmation response from the subscriber. During the specified time frame if the message is not received, the message is been re-sent. Using this method the subscriber may receive the same message more than once if the broker does not receive the subscriber's acknowledgment in time. This process is referred to as "at least one delivery". [13]

QoS 2 - The client and broker use a 4-step integration process to ensure that the message is received, and it has been received only once. This is also referred to as "Exactly once delivery". [13]

For best communication systems QoS 0 is the best option, where the communications are reliable and are limited. But where the communications are unreliable and resource connections are not limited, then QoS - 2 is the best option. Using QoS - 1 provides a sort of best-of-both-worlds solution but requires that the application receiving the data knows how to handle duplicates. [13]

### 2.5.2 MQTT Architecture

MQTT protocol runs on a TCP/IP using a PUBLISH/SUBSCRIBE model. In this protocol, there are two types of systems- Clients and brokers. A broker is a server that the clients communicate with. The broker receives the communications from the clients and sends those communications to other clients. But Clients do not interact directly with each other but connect to the broker. Each client can be a Publisher or a subscriber. A client only publishes

when there is information to be sent, and the broker only sends out information to subscribers when the new data arrives.

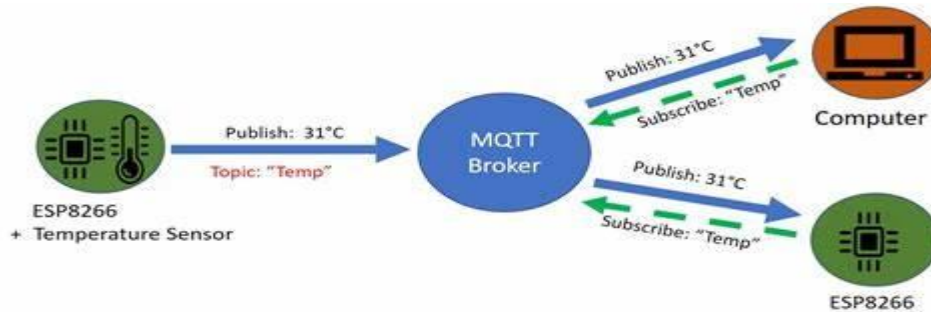


Figure 18 MQTT Architecture[14]

Diagram understanding about how MQTT protocol works with an example of ESP8266 Sensor.

### 2.5.3 MQTT Broker

MQTT Broker is a central software entity in the MQTT architecture. It is a lightweight protocol that supports the IoT. MQTT is an intermediary entity that enables MQTT clients to communicate. The broker receives messages published by clients, filters the messages by topic, and distributes them to subscribers. Using MQTT brokers to enable the Publish/Subscriber model helps make MQTT a highly efficient and scalable protocol. [14]

There are two types of brokers:

1. Managed Brokers
2. Self - Hosted Brokers

**Managed Broker** - This doesn't require the user to set up anything on your server to enable MQTT communication. Managed broker services allow the user to host their hosted brokers in their systems. The best example is - AWS IoT Core.

Self-Hosted Broker - As the name suggests self - hosted broker requires the user to install the broker on the user's server with a static IP. The installation process is quite user-friendly and is not difficult in managing, securing, and scaling the brokers. There are many open-source implementations of MQTT brokers like - Mosquitto, hivemq. [14]

#### 2.5.4 Mosquitto Broker

Eclipse Mosquitto is an open-source (EPL/EDL) message broker that is implemented in the MQTT protocol in limited versions. Mosquitto is lightweight and is suitable for use on all devices from low-power single-board computers to full servers. MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it more suitable for Internet of Things messaging such as low-power sensors, mobile devices, and microcontrollers. The mosquitto broker also provides a C library for implementing MQTT clients and the most popular mosquitto\_pub command line for MQTT clients. [15]



Figure 19 Mosquitto Broker [15]

#### 2.6 Node-Red

Node-Red is a programming tool for writing together hardware devices, and API's in new and different ways. It provides a browser-based editor that makes it quite efficient to wire together flows using the wide range of nodes and diodes in the palette that can be deployed to its runtime in a single-click. [17]

Node-RED provides a flow editor-based editor that makes it quite easy to work and understand the workings. Flows can then be deployed to the run time in a single click. A built-in library allows you to save useful functions, templates, or flows for re-use. Javascript functions can be created within the editor using a rich text editor. The flow created in Node-RED is stored using JSON which can be easily imported and exported for sharing with others. An online flow library allows the user to share the flows. [18]

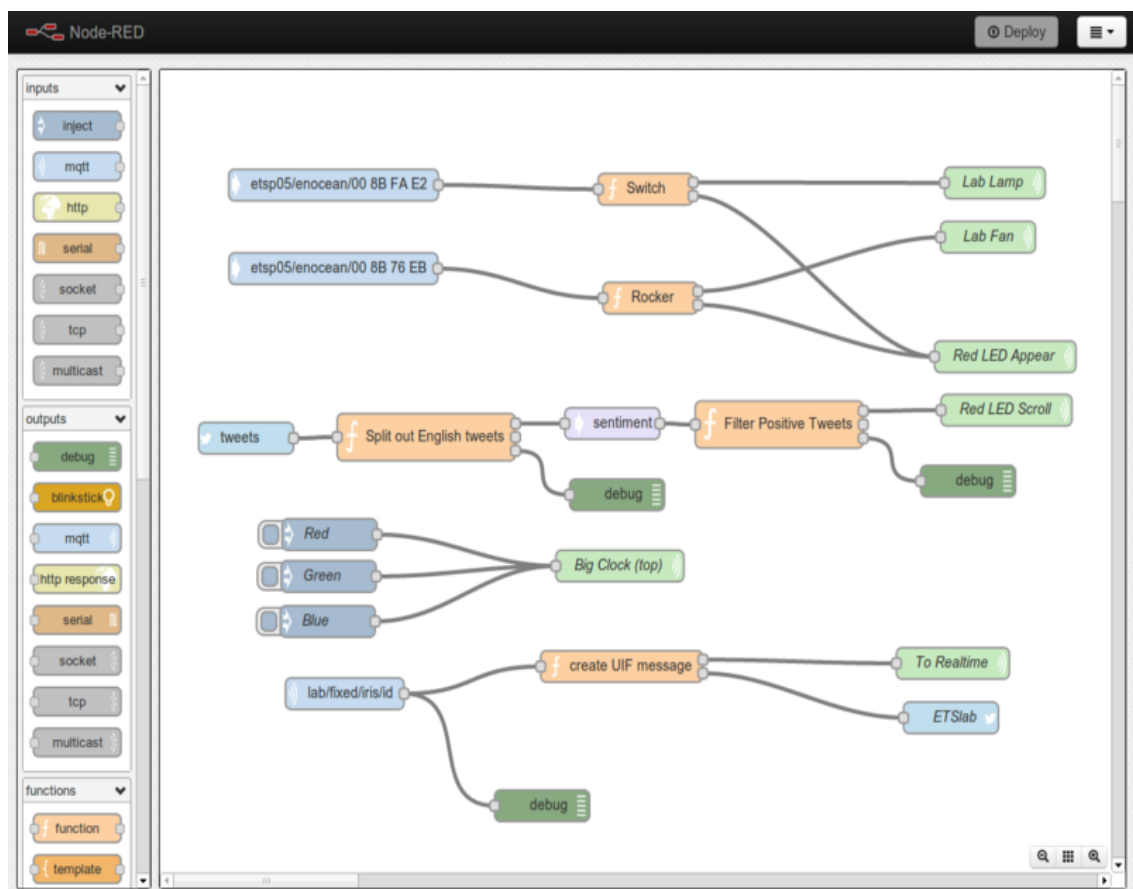


Figure 20 Node-RED Working [18]

## 2.7 PuTTY Software

PuTTY is a free open-source software for Windows devices that gives users the option to transfer data securely and safely. The client uses different kinds of file transfer protocols such as SCP, SSH, and Rlogin to encrypt data and protect it from unauthorized use. The tool comes up with a Command Line Interface for

manipulating important functions. This program has the abilities of an SSH terminal so that users can easily establish a connection and send data safely, it also provides a Key Authentication. [19]

This software allows the users to connect to switches, mainframes, and servers via SSH and serial clients. This software is quite handy, especially when working using the public internet. It tunnels user sessions using different clients like PSCP, and PSFTP. [19]

This program uses various protocols to ensure data doesn't fall into the hands of hackers and is transferred to its destination without any interruptions. It's quite easy to configure this for network connections, we can start specifying the type of connection, destination, log file, and other details. The software offers various settings that give better control over it. As a terminal emulator, the program allows the user to host different operating systems on the device. Moreover, the software is quite useful in running Android, Unix, Mac, and IOS. [19]

It is free and open-source software is quite reliable and supports a wide range of protocols for secure sharing and connections. PuTTY software offers many features which make the whole process of working becomes easy.

### 3 Implementation

The project is initiated from the hardware side, where publishing BME688 sensor readings (temperature, gas, humidity, pressure, and air quality) via MQTT with the ESP32. All the published sensor readings will be displayed in the Node-RED Dashboard as it supports the MQTT platform. The ESP32 is programmed in Arduino IDE. MQTT protocol and the MQTT Mosquitto broker are installed on the Raspberry Pi board. An alternative way, if neglecting the Raspberry Pi then download the MQTT broker on your computer, for example, Cloud Broker.

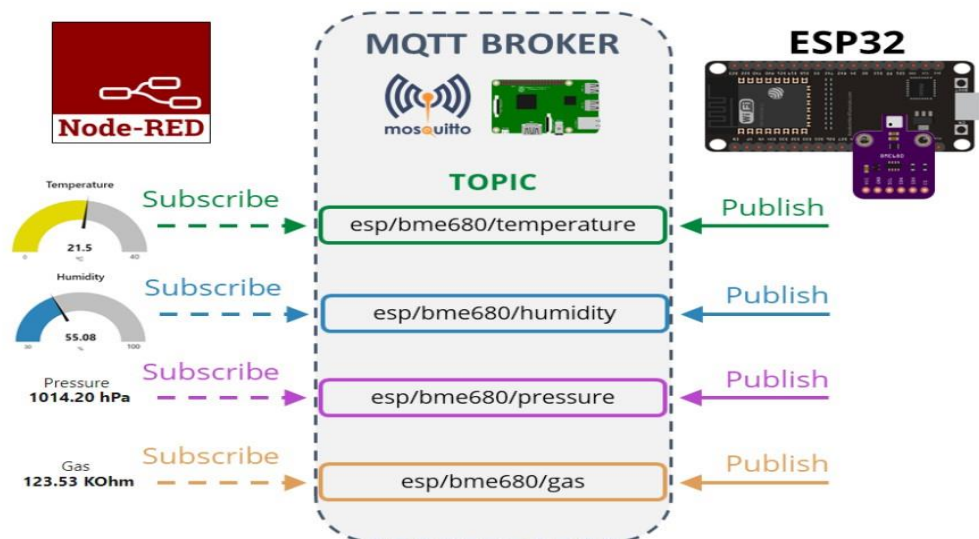


Figure 21 Thesis Project Idea

This figure best describes the project and working of MQTT protocol. The figure contains BME680 sensor which is same as BME688 sensor both belong to the same family of sensors.

### 3.1 Schematic Design of the Project

Discussing more in detail regarding the circuit of this project -

The GND pin in ESP32 is connected to the GND pin in the BME688 sensor. The GND pin is for the output of an onboard voltage regulator. The pin can be used to power supply the external components. GND is a ground pin of the ESP32 development board.

The GPIO 22 pin in ESP32 is connected to the SCK pin in the BME688 sensor. The ESP32 comes up with 2 internal I2C channels and also a default pair of pins for the I2C communication protocol. The default I2C pins of ESP32 are - SDA(GPIO 21) and SCL(GPIO 22). The SCK pin in the BME688 sensor is Serial Clock, it sunders the SPI interface which is compatible with the SPI mode "00" (CPOL = CPHA = "0") and mode "11" (CPOL = CPHA = "1"). The automatic selection between the "00" and "11" is determined by the value of SCK after the CSB falling edge.

The GPIO pin 21 in ESP32 is connected to the SDI of the BME688. As discussed earlier the GPIO 21 PIN is the default I2C pin in the ESP32. While the SDI in BME688 is the Serial Data Input, the data input/ output is in 3 wire mode.

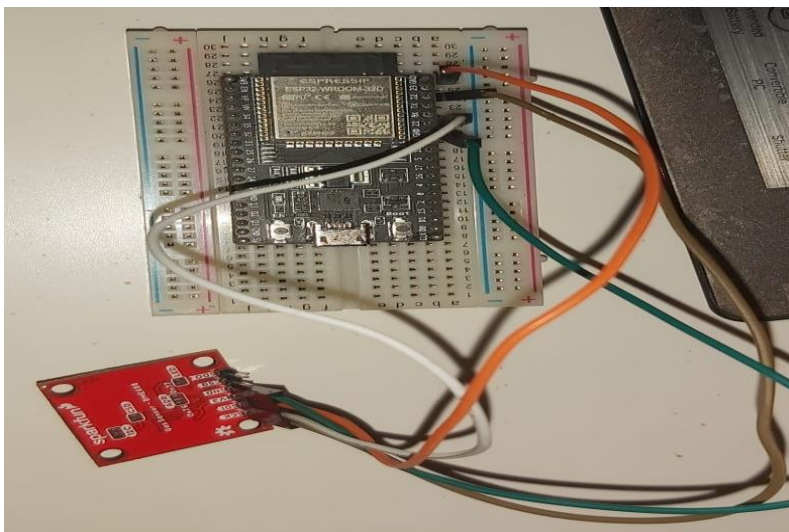


Figure 22 Working Circuit



### 3.2 Working details on software side

While working with the software side of the project there are some libraries and software to be installed before starting to work on this project.

Firstly, the ESP32 add-on library must be installed. In Arduino IDE go to Files > Preferences.

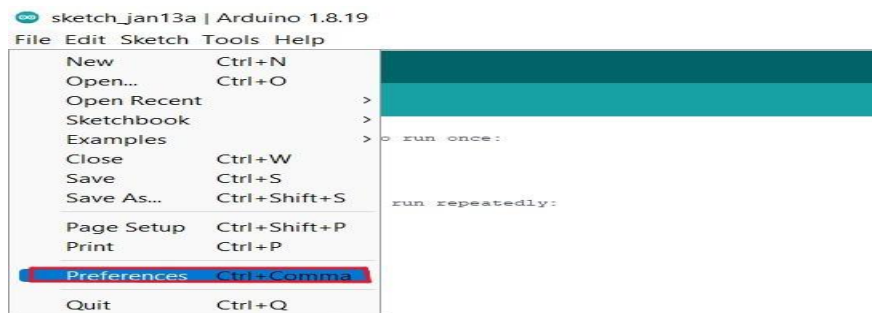


Figure 23 Clicking add on library to install ESP32

Then Enter the following URLs into the Additional Board Manager

[https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json)

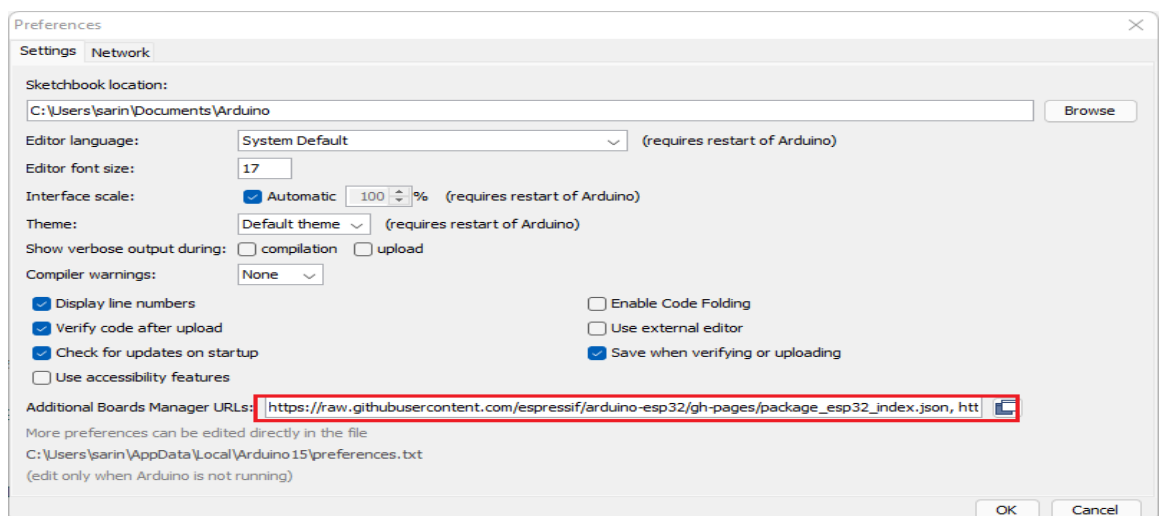


Figure 24 Inserting a URL for downloading the ESP32

Click OK! button.

Open the Board Manager and go to Tools > Boards > Board Manager

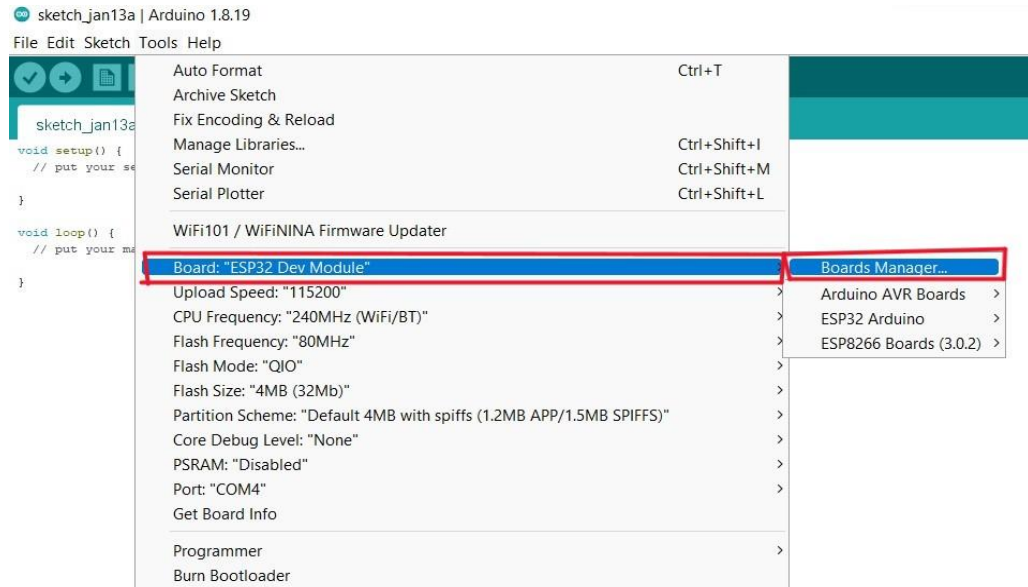


Figure 25 Selecting board from Board Manager

Search for the ESP32 and install it by "Espressif Systems".

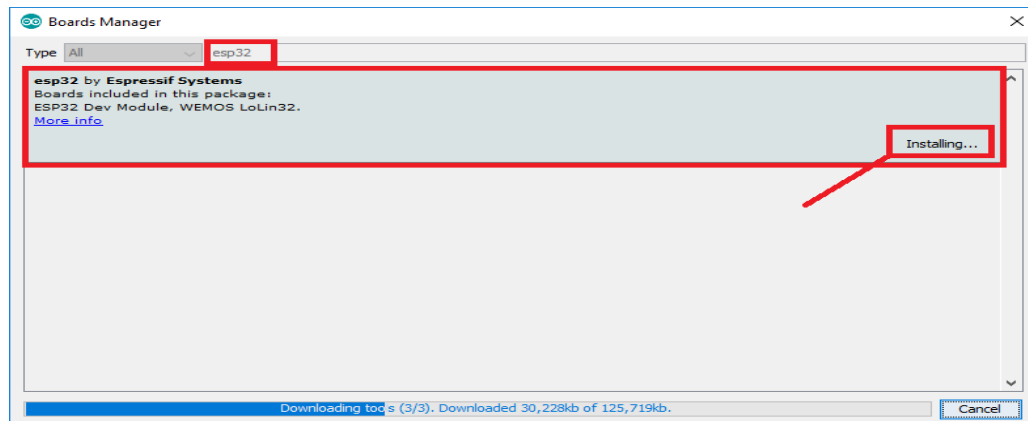


Figure 26 Installing ESP32 by Espressif Systems

It might take a few seconds to install.

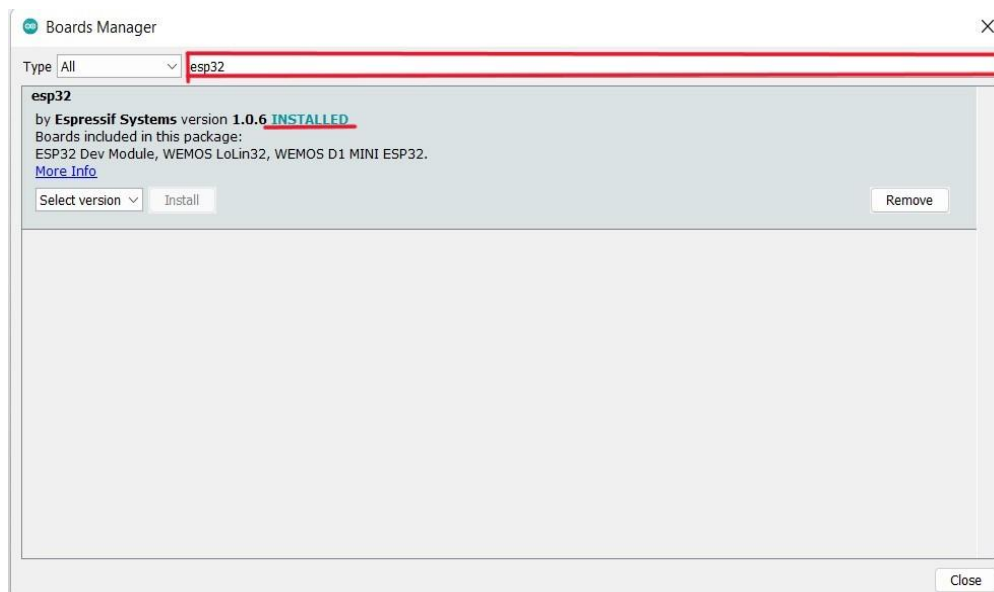


Figure 27 ESP32 Installed

## Testing the installation process

Go to Tools > Boards > (In my system it shows "DOIT ESP32 DEVKIT V1").

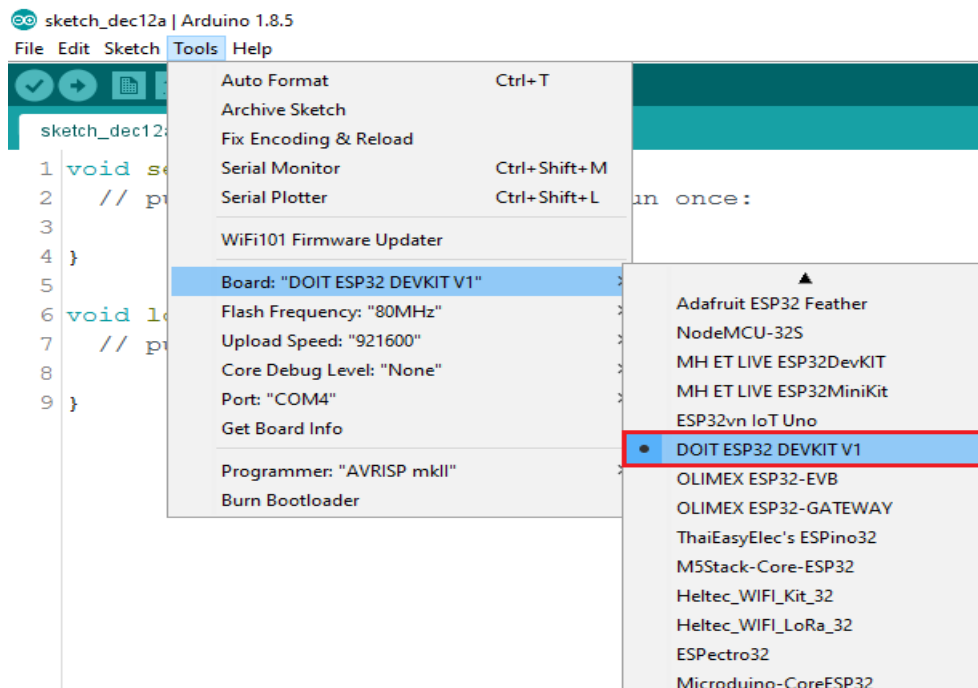


Figure 28 Selecting board for testing the installation of ESP32

Select the COM port in your Arduino IDE.

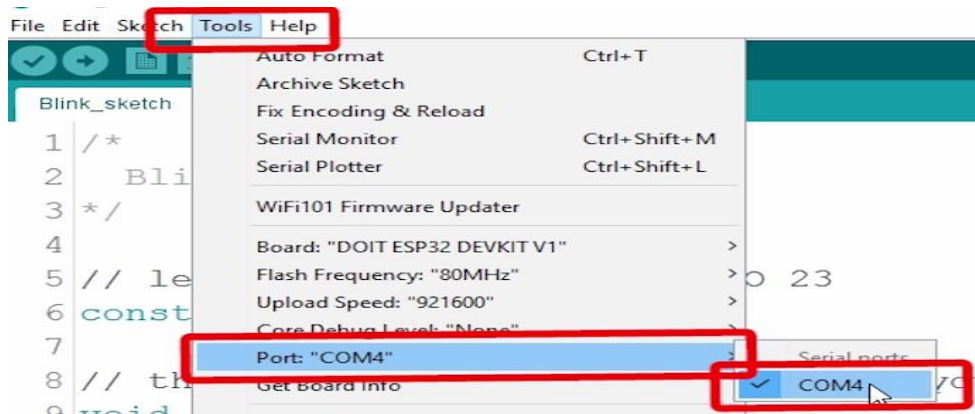


Figure 29 Selecting COM4 port

Open the example by File > Examples > WiFi(ESP32) > WiFiScan

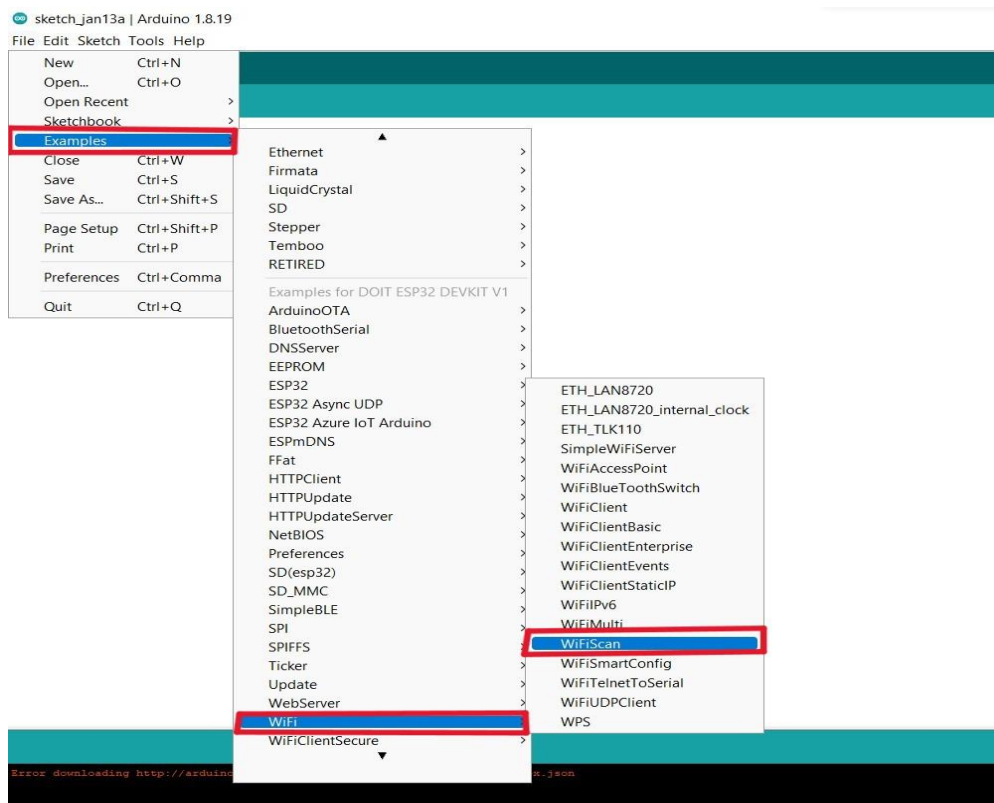


Figure 30 Selecting WiFi Modules

A new Sketch window opens up in the Arduino IDE.



```

WiFiScan | Arduino 1.8.19
File Edit Sketch Tools Help

WiFiScan

/*
 * This sketch demonstrates how to scan WiFi networks.
 * The API is almost the same as with the WiFi Shield library,
 * the most obvious difference being the different file you need to include:
 */
#include "WiFi.h"

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

  // Set WiFi to station mode and disconnect from an AP if it was previously connected
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");
}

void loop()
{
  Serial.println("scan start");

  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();
  Serial.println("scan done");
  if (n == 0) {
    Serial.println("no networks found");
  } else {
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
      delay(10);
    }
  }
  Serial.println("");

  // Wait a bit before scanning again
  delay(5000);
}

```

Figure 31 New Sketch windows of WiFiScan

After the sketch opens press the Upload button, and the IDE compiles and uploads the board.



If everything went as expected then the following "Done Uploading" message would appear.

```
Done uploading.
Writing at 0x00042000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds
Hash of data verified.

Leaving...
Hard resetting...

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM4
```

Figure 32 Done Uploading

Open the serial monitor at a baud rate of 115200.



Press the ESP32 on board Enable button and see the networks available near the ESP32.

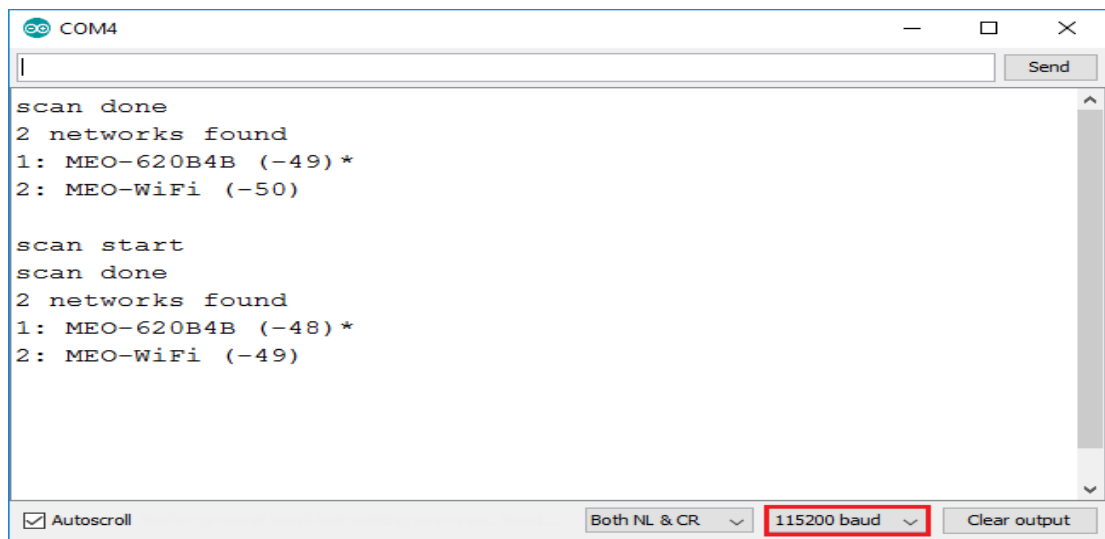


Figure 33 Successful working of ESP32 in COM4 Port

Installing Mosquitto Broker on Raspberry Pi OS with the help of PuTTY software.

Before installing raspberry pi in windows, an SSH connection should be established between the computer and the raspberry pi. SSH ( Secure Shell), is a method of establishing communication with another computer securely. All data sent via SSH is encrypted and it is based on a Unix Shell, so it allows the raspberry pi files to remote machine by using terminal commands. This project is been done on the Windows platform so PuTTY is the best choice to work with.

To download the PuTTY software browse through [www.putty.org](http://www.putty.org)

Download PuTTY it is recommended to download the **putty.exe file**. Run the putty.exe file to execute the software.

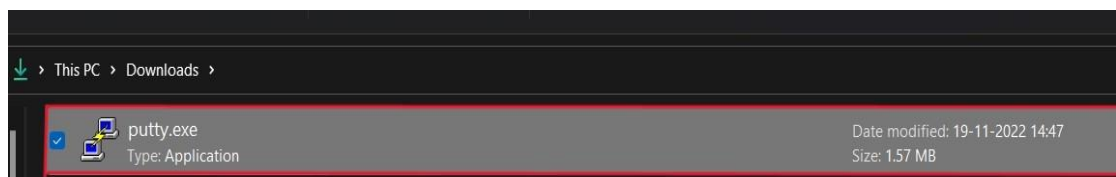


Figure 34 PuTTY software download

Connecting to the Raspberry Pi via SSH

When the PuTTY software is installed, power on the Raspberry Pi and follow the procedure-

1. Open PuTTY
2. Select the options -

Host Name - raspberrypi (It is a default hostname).

Port 22

Connection type: SSH

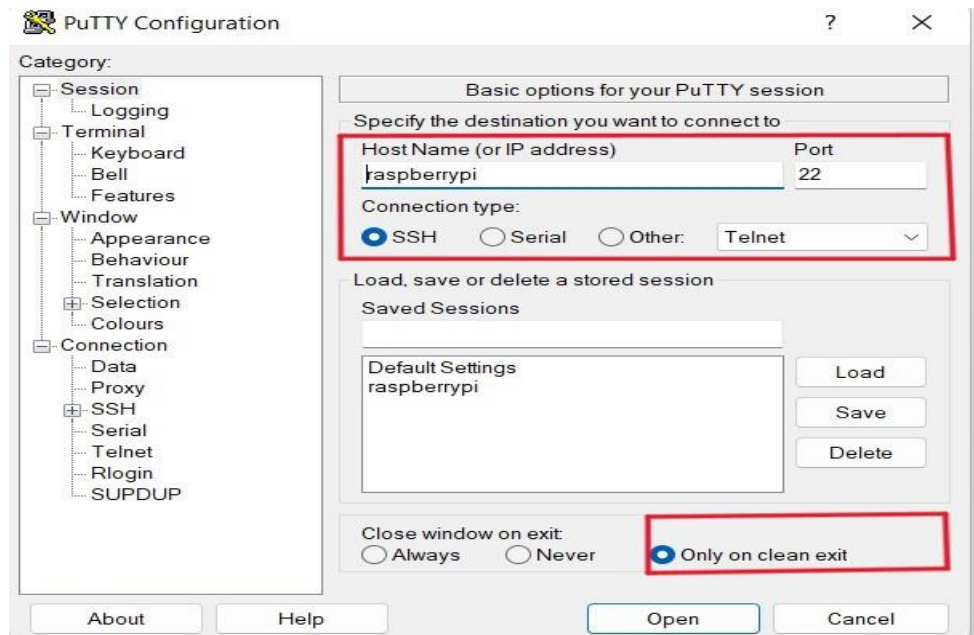


Figure 35 Selecting hostname and connection type in PuTTY software

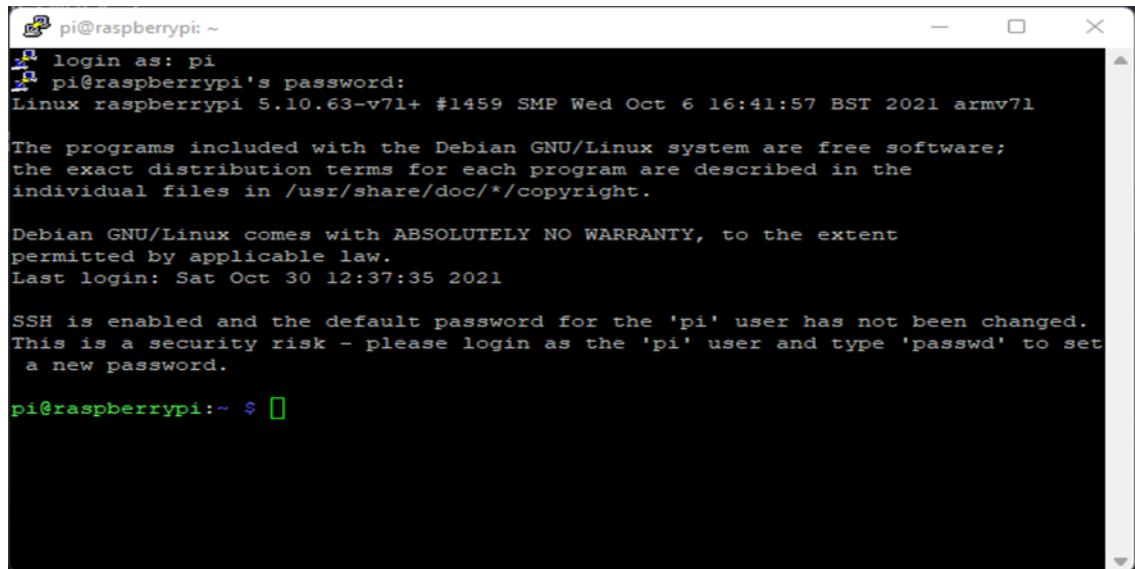
Click Open

When using it for the first time it might prompt a warning message, just ignore it and simply Click, NO.

Now, the user has to enter the login credentials of the Raspberry Pi to set the installation process.

In the new window type the username and hit Enter. While entering your password, the characters won't be showing up.





```

pi@raspberrypi: ~
login as: pi
pi@raspberrypi's password:
Linux raspberrypi 5.10.63-v7l+ #1459 SMP Wed Oct 6 16:41:57 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Oct 30 12:37:35 2021

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $ █

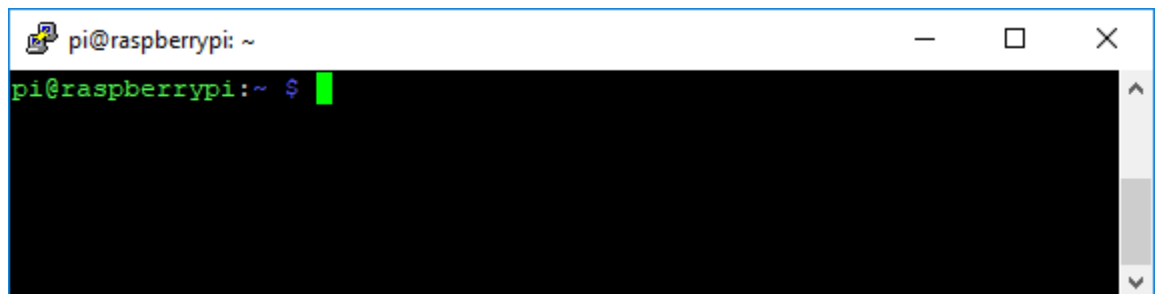
```

Figure 36 Logging in as pi and password

After feeding your username and password there is a successfully established SSH connection with the Raspberry Pi.

Now, coming back to installing the MQTT Mosquitto broker.

Open the Raspberry Pi terminal window.



```

pi@raspberrypi: ~
pi@raspberrypi:~ $ █

```

Figure 37 Opening of the Terminal window of PuTTY

Run the following command to upgrade the system:

```
sudo apt update && sudo apt upgrade
```

After writing the command, Press Y and Enter it might take some time to update.

For installing Mosquitto type the following command-

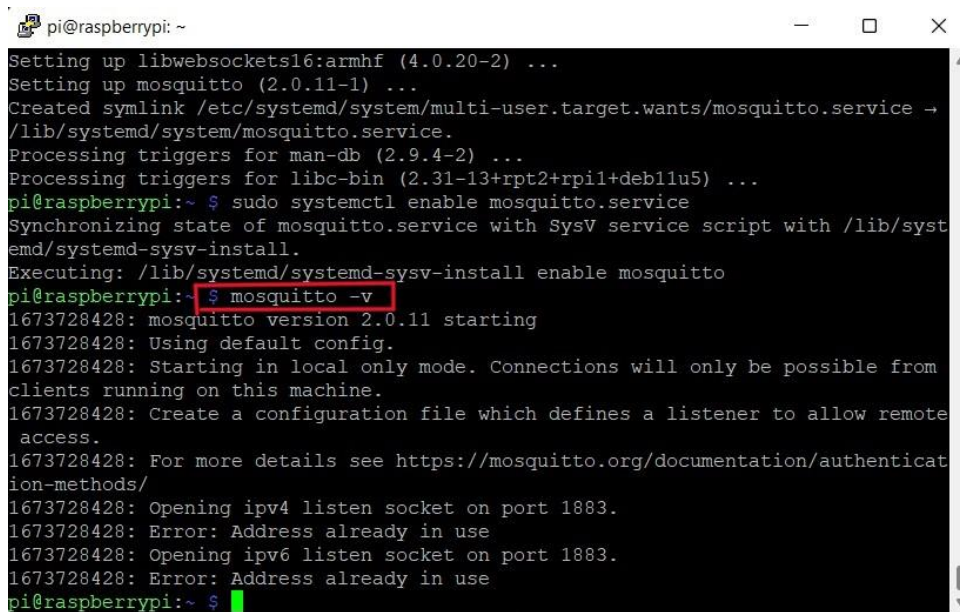
```
sudo apt install -y mosquitto mosquitto-clients
```

To make the MQTT Mosquitto Broker auto start bot the RaspberryPi and run the following command

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

With this, test the installation by running a simple command.

```
mosquitto -v
```



```

pi@raspberrypi: ~
Setting up libwebsockets16:armhf (4.0.20-2) ...
Setting up mosquitto (2.0.11-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/mosquitto.service →
/lib/systemd/system/mosquitto.service.
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+rpt2+rpil+deb11u5) ...
pi@raspberrypi:~ $ sudo systemctl enable mosquitto.service
Synchronizing state of mosquitto.service with SysV service script with /lib/syst
emd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
pi@raspberrypi:~ $ mosquitto -v
1673728428: mosquitto version 2.0.11 starting
1673728428: Using default config.
1673728428: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1673728428: Create a configuration file which defines a listener to allow remote
access.
1673728428: For more details see https://mosquitto.org/documentation/authenticat
ion-methods/
1673728428: Opening ipv4 listen socket on port 1883.
1673728428: Error: Address already in use
1673728428: Opening ipv6 listen socket on port 1883.
1673728428: Error: Address already in use
pi@raspberrypi:~ $

```

Figure 38 Mosquitto Broker version command

It will show the latest version of the Mosquitto broker is been installed.

That's how the MQTT Mosquitto broker is installed in Raspberry Pi.

### Raspberry Pi IP Address

To use the Mosquitto broker for future projects, having Raspberry Pi IP address is important, from this retrieving of data can be possible. To find the IP address type the following command-

```
hostname -I
```

```
1673720420: ERROR: Address already in use
pi@raspberrypi:~ $ hostname -I
169.254.226.168 192.168.0.100
pi@raspberrypi:~ $
```

Figure 39 Hostname to get the Raspberry Pi IP address

In this case, the Raspberry Pi IP address is 192.168.0.100. It's better to save it for future projects.

Now to see the data Node-RED dashboard comes into the picture, Installing Node-Red dashboard on Raspberry Pi. Node-RED is a powerful open-source tool for visual programming to build IoT applications. Node-RED runs on the web browser and it uses visual programming that allows the user to connect code blocks, which are known as Nodes. The nodes when wired together are called Flows.

As established an SSH connection before with the RaspberryPi installing Node-RED would be easy by following the command:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

Node-RED is installed by default on the Raspberry Pi OS(32-Bit). While installing there would be few messages informing the user, just simply enter Y and Enter to accept. The following messages will be to install Pi-specific nodes- Press Y and Enter. Usually, it takes a few minutes to install Node-RED, in the end, a similar window might be on the terminal window of the user.

```

pi@raspberrypi: Node-RED update

Running Node-RED install for user pi at /home/pi on raspbian

This can take 20-30 minutes on the slower Pi versions - please wait.

Stop Node-RED
Remove old version of Node-RED
Remove old version of Node.js
Install Node.js 16 LTS          v16.19.0   Npm 8.19.3
Clean npm cache
Install Node-RED core          3.0.2
Move global nodes to local
Npm rebuild existing nodes
Install extra Pi nodes
Add shortcut commands
Update systemd script

Any errors will be logged to /var/log/nodered-install.log
All done.
You can now start Node-RED with the command node-red-start
or using the icon under Menu / Programming / Node-RED
Then point your browser to localhost:1880 or http://{your_pi_ip-address}:1880

Started : Sat 14 Jan 20:58:16 GMT 2023
Finished: Sat 14 Jan 21:00:11 GMT 2023

*****
**

```

Figure 40 Installation of necessary modules of Node-RED

After installing it is recommended to configure initial options and settings by entering the following command:

```
node-red admin init
```

```

would you like to customise the settings now (y/n) : n
pi@raspberrypi:~ $ node-red admin init
Node-RED Settings File initialisation
=====
This tool will help you create a Node-RED settings file.

  Settings file · /home/pi/.node-red/settings.js

User Security
=====
  Do you want to setup user security? · Yes
  Username · raspberrypi
  Password · *****
  User permissions · full access
  Add another user? · No

Projects
=====
The Projects feature allows you to version control your flow using a local git repository.

  Do you want to enable the Projects feature? · No

Flow File settings
=====
  Enter a name for your flows file · flows.json
  Provide a passphrase to encrypt your credentials file ·

Editor settings
=====
  Select a theme for the editor. To use any theme other than "default", you will need to install @node-red-contrib-themes/theme-collection in your Node-RED user directory. · default

  Select the text editor component to use in the Node-RED Editor · monaco (default)

Node settings
=====
  Allow Function nodes to load external modules? (functionExternalModules) · Yes

Settings file written to /home/pi/.node-red/settings.js
pi@raspberrypi:~ $ █

```

Figure 41 Implementations of Sequence for Node-RED

After that, there would be a sequence of questions regarding how to implement, security, file address, the theme of the editor, and installation of external modules.

Now, after the configuration of the Node-RED is completed, all the settings are saved on **settings.js**

The start of the Node-RED is done by running the following command:

## Program 8. Descriptive Name

node-red-start

```

pi@raspberrypi:~ $ node-red-start
Start Node-RED

Once Node-RED has started, point a browser at http://169.254.226.168:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use node-red-stop          to stop Node-RED
Use node-red-start        to start Node-RED again
Use node-red-log          to view the recent log output
Use sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
14 Jan 21:09:13 - [info]
Welcome to Node-RED
=====
14 Jan 21:09:13 - [info] Node-RED version: v3.0.2
14 Jan 21:09:13 - [info] Node.js version: v16.19.0
14 Jan 21:09:13 - [info] Linux 5.15.61-v7l+ arm LE
14 Jan 21:09:14 - [info] Loading palette nodes
14 Jan 21:09:16 - [info] Settings file : /home/pi/.node-red/settings.js
14 Jan 21:09:16 - [info] Context store : 'default' [module=memory]
14 Jan 21:09:16 - [info] User directory : /home/pi/.node-red
14 Jan 21:09:16 - [warn] Projects disabled : editorTheme.projects.enabled=false
14 Jan 21:09:16 - [info] Flows file : /home/pi/.node-red/flows.json
14 Jan 21:09:16 - [info] Creating new flow file
14 Jan 21:09:16 - [warn] Using unencrypted credentials
14 Jan 21:09:16 - [info] Server now running at http://127.0.0.1:1880/
14 Jan 21:09:16 - [info] Starting flows
14 Jan 21:09:16 - [info] Started flows

```

Figure 42 Node-RED Start

### Autostarting Node-RED on boot

To automatically run the Node-RED when the Pi boots up, run the following command. In this case, as long as the Raspberry Pi is powered on Node-RED will be running. This step saves a lot of time to restart the Node-RED.

```
sudo systemctl enable nodered.service
```

The access of the Node-RED dashboard can be run on port 1880. To access the Node-RED open a browser and type the Raspberry Pi IP address followed by 1880. In this project the IP address used is :

192.168.0.100:1880

To get the Raspberry Pi IP address type the following command:

```
hostname -I
```

```
Last login: Sat Jan 14 20:56:17 2016
pi@raspberrypi:~$ hostname -I
169.254.226.168 192.168.0.100
pi@raspberrypi:~$
```

Figure 43 IP Address

After entering the RaspberryPi IP address followed by:1880 on the web browser, the user gets access to the Node-Red login page.

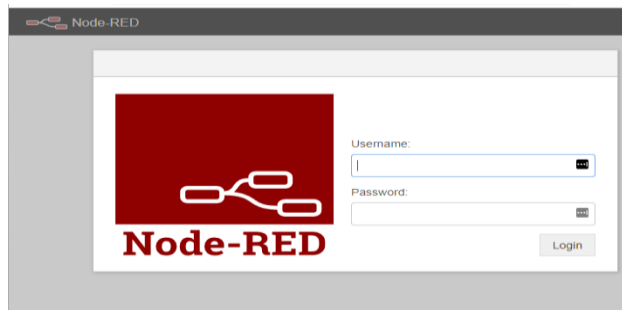


Figure 44 Node-RED login page

These were the process of installing Node-RED dashboard on Raspberry Pi.

```

#define WIFI_SSID "*****"
#define WIFI_PASSWORD "*****"

// Raspberry Pi Mosquitto MQTT Broker
#define MQTT_HOST IPAddress(192, 168, 0, 100)
// For a cloud MQTT broker, type the domain name
//#define MQTT_HOST "example.com"
#define MQTT_PORT 1883

// Temperature MQTT Topics
#define MQTT_PUB_TEMP "esp/bme688/temperature"
#define MQTT_PUB_HUM "esp/bme688/humidity"
float gasResistance;

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMil
#define MQTT_PUB_PRES "esp/bme688/pressure"
#define MQTT_PUB_GAS "esp/bme688/gas"

```

Figure 45 Connection of Raspberry Pi and MQTT broker

```

// Variables to hold sensor readings
float temperature;
float humidity;
float pressure;lis = 0; // Stores last time temperature was published
const long interval = 10000; // Interval at which to publish sensor readings

void getBME688Readings(){
  // Tell BME688 to begin measurement.
  unsigned long endTime = bme.beginReading();
  if (endTime == 0) {
    Serial.println(F("Failed to begin reading :("));
    return;
  }
  if (!bme.endReading()) {
    Serial.println(F("Failed to complete reading :("));
    return;
  }
  temperature = bme.temperature;
  pressure = bme.pressure / 100.0;
  humidity = bme.humidity;
  gasResistance = bme.gas_resistance / 1000.0;
}

```

Figure 46 Getting BME688 readings



```
void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
  Serial.println("Connecting to MQTT...");
  mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
  Serial.printf("[WiFi-event] event: %d\n", event);
  switch(event) {
    case SYSTEM_EVENT_STA_GOT_IP:
      Serial.println("WiFi connected");
      Serial.println("IP address: ");
      Serial.println(WiFi.localIP());
      connectToMqtt();
      break;
    case SYSTEM_EVENT_STA_DISCONNECTED:
      Serial.println("WiFi lost connection");
      xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT while reconnecting to Wi-Fi
      xTimerStart(wifiReconnectTimer, 0);
      break;
  }
}
```

Figure 47 Connection of the WiFi

```

void loop() {
  unsigned long currentMillis = millis();
  // Every X number of seconds (interval = 10 seconds)
  // it publishes a new MQTT message
  if (currentMillis - previousMillis >= interval) {
    // Save the last time a new reading was published
    previousMillis = currentMillis;

    getBME688Readings();
    Serial.println();
    Serial.printf("Temperature = %.2f °C \n", temperature);
    Serial.printf("Humidity = %.2f % \n", humidity);
    Serial.printf("Pressure = %.2f hPa \n", pressure);
    Serial.printf("Gas Resistance = %.2f KOhm \n", gasResistance);

    // Publish an MQTT message on topic esp/bme688/temperature
    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true, String(temperature).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_TEMP, packetIdPub1);
    Serial.printf("Message: %.2f \n", temperature);

    // Publish an MQTT message on topic esp/bme688/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true, String(humidity).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", humidity);

    // Publish an MQTT message on topic esp/bme688/pressure
    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true, String(pressure).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_PRES, packetIdPub3);
    Serial.printf("Message: %.2f \n", pressure);

    // Publish an MQTT message on topic esp/bme688/gas
    uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_GAS, 1, true, String(gasResistance).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_GAS, packetIdPub4);
    Serial.printf("Message: %.2f \n", gasResistance);
  }
}

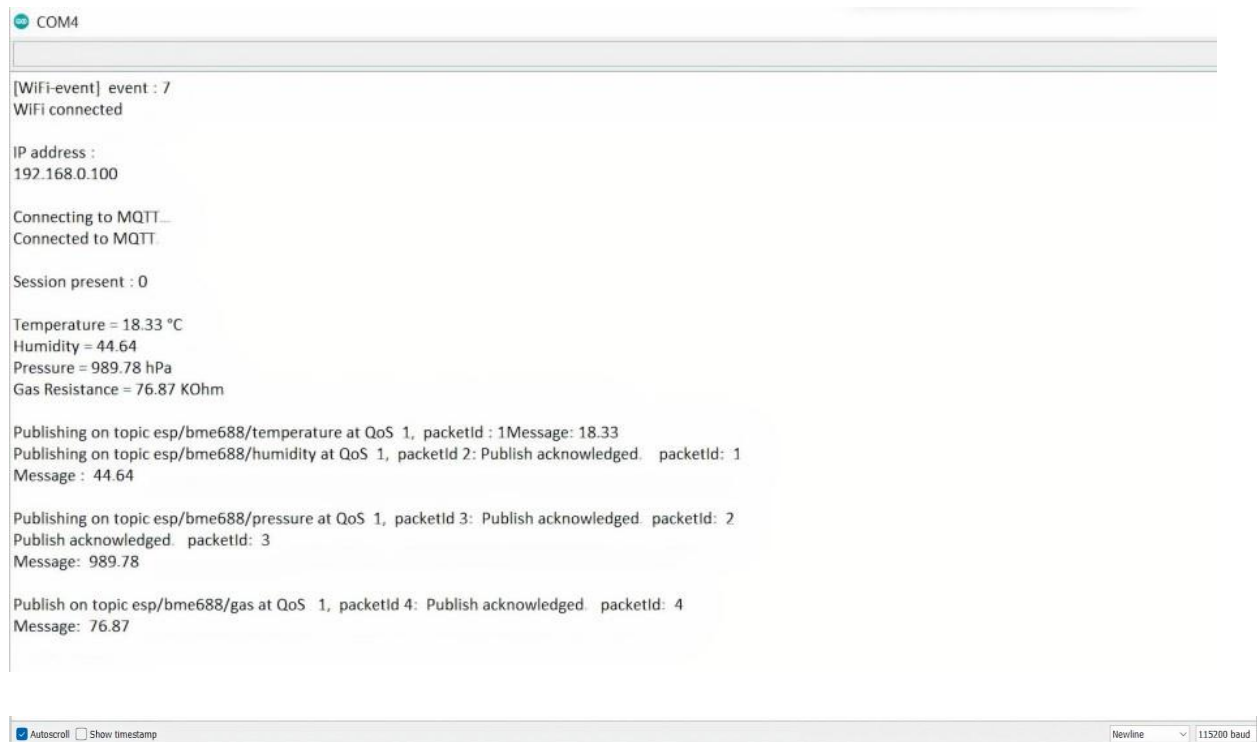
```

Figure 48 Getting readings from BME688 after regular interval

In the above mentioned of the codes from the project symbolises the fact of how the MQTT broker, Node-RED is been installed in the Raspberry Pi with an established IP address. Following to it in the code all the measurements are been specified to get an adequate data with correct scientific units. The code also describes the connection of the WiFi with the BME688 sensor, where the connection can be established and disconnected. In the last part of the code , Is the section were the code is programmed to get all the readings from the sensor.Publishing of the data in the Node-RED dashboard to get a proper visual of the sensed data.

## 4 RESULT

After the completion of the thesis project, the goal is achieved by showing the data communication using MQTT with the BME688 sensor. Below in figure 49, the observation can be made that the BME688 sensor is sensing the data accurately and publishing it in the serial monitor of the Arduino IDE. Where the sensor first connects to the Wi-Fi and then shares the IP address through which we can access the node-Red dashboard for better data visualization. Following it, the temperature, humidity, pressure, and gas data can be shown and lastly, the MQTT acknowledges the publishing of the data.



```
COM4

[WiFi-event] event : 7
WiFi connected

IP address :
192.168.0.100

Connecting to MQTT...
Connected to MQTT.

Session present : 0

Temperature = 18.33 °C
Humidity = 44.64
Pressure = 989.78 hPa
Gas Resistance = 76.87 KOhm

Publishing on topic esp/bme688/temperature at QoS 1, packetId : 1Message: 18.33
Publishing on topic esp/bme688/humidity at QoS 1, packetId 2: Publish acknowledged. packetId: 1
Message : 44.64

Publishing on topic esp/bme688/pressure at QoS 1, packetId 3: Publish acknowledged. packetId: 2
Publish acknowledged. packetId: 3
Message: 989.78

Publish on topic esp/bme688/gas at QoS 1, packetId 4: Publish acknowledged. packetId: 4
Message: 76.87

Autoscroll  Show timestamp Newline  115200 baud
```

Figure 49 Serial monitor result published

In figure 50 observations can be made regarding how the connection is made and showing how the Node-RED dashboard virtually connects the components and subscribes back the data to the user.

Figure 50 Node-RED Connection

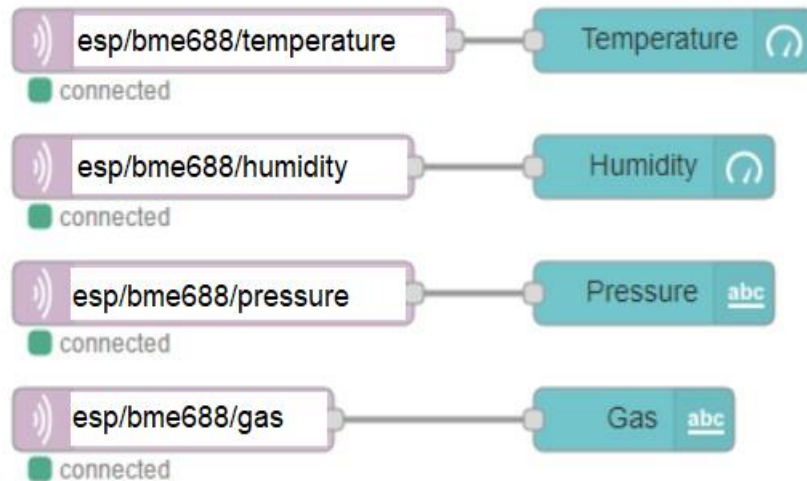


Figure 50 Node-RED Connection

Lastly, a distinct image of the complete circuit which is been in use for the implementation of the project. The ESP32 is powered on and the connection with the BME688 sensor with the specific pins through which all the data is been sensed is the goal of the thesis project. Raspberry Pi is also in action where it is responsible for the installations of the software which is the Node-RED and Mosquito Broker.

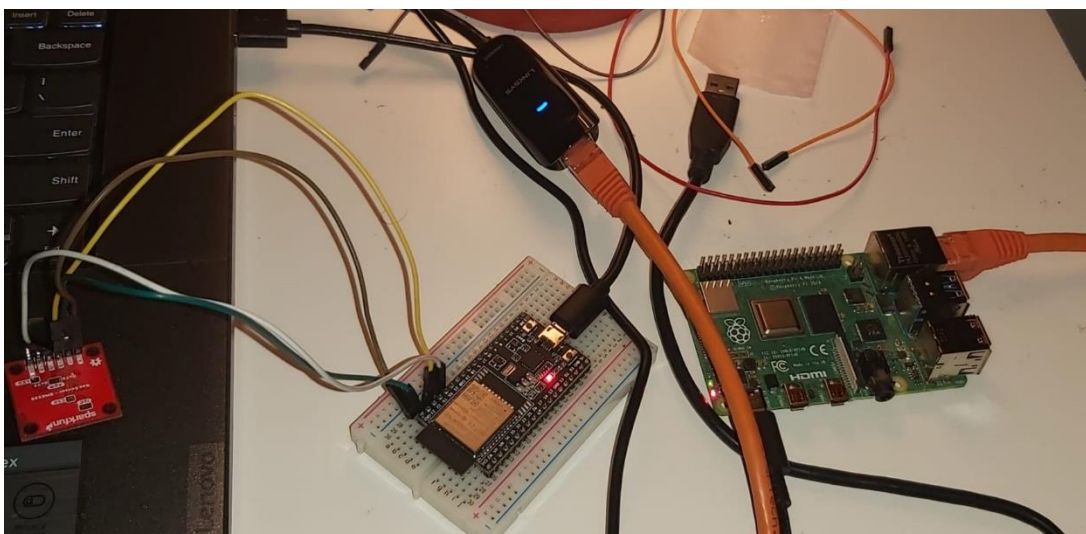


Figure 51 Circuit implementation

## 5 CONCLUSION

The main idea of this thesis is to show the data communication using MQTT with BME688 sensor Gas AI sensor developed by BOSCH. The BME688 sensor is so well equipped that sensing gas, pressure, humidity and air quality data becomes possible to achieve. Transmitting the data from the sensor with ESP32 microcontroller programmed in Arduino IDE, as installing all the necessary libraries and modules becomes easy to handle. The thesis project is obeying a certain and most important protocol, MQTT- which is mainly used in IoT projects and also helps the messages to be encrypted as the data transmitting is not in high quantity. After all the data has been sensed from the real environment, it is displayed in the Node-RED dashboard. Node-RED is a very powerful open-source for developing IoT applications. It runs on a web browser and uses a virtual programming experience which allows the user to connect with the nodes. The MQTT Broker(Mosquitto) is installed in Raspberry Pi and also the Node-RED dashboard. The results achieved is also success , the BME688 was able to send the data. To communicate with the Raspberry Pi via SSH PuTTY software was been implemented to install, PuTTY uses SSH connection as a method of establishing a communication with another computer more securely. All the data sent via SSH encrypted and also PuTTY also allows to access the Raspberry Pi files from a remote machine by terminal commands.

## References

1. www.tutorialspoint.com. (n.d.). *Introduction to ESP32*. [online] Available at:  
[https://www.tutorialspoint.com/esp32\\_for\\_iot/esp32\\_for\\_iot\\_introduction.htm#](https://www.tutorialspoint.com/esp32_for_iot/esp32_for_iot_introduction.htm#)  
[Accessed 20 Dec. 2022].
2. Electronicshub.org. (2023). [online] Available at:  
<https://www.electronicshub.org/wp-content/uploads/2021/02/ESP32-Pinout-1.jpg>  
[Accessed 20 Dec. 2022].
3. Electropeak. (2021). *Full Guide to ESP32 Pinout Reference: What GPIO Pins Should We Use?* [online] Available at: <https://electropeak.com/learn/full-guide-to-esp32-pinout-reference-what-gpio-pins-should-we-use/#:~:text=The%20ESP32%20comes%20with%20%20internal%20IC%20channels>  
[Accessed 21 Dec. 2022].
4. www.raspberrypi.com. (n.d.). *Raspberry Pi Documentation - Computers*. [online] Available at:  
<https://www.raspberrypi.com/documentation/computers/>.
5. Pi Day. (n.d.). *Applications of Raspberry Pi*. [online] Available at:  
<https://www.piday.org/interesting-applications-of-raspberry-pi/>  
[Accessed 22 Dec. 2022].
6. Admin (2022). *Raspberry pi pico pinout diagram datasheet specifications*. [online] Available at: <https://mechtrician.com/raspberry-pi-pico-pinout-diagram-datasheet-specifications/#:~:text=Raspberry%20pi%20pico%20pinout%20diagram%20%28source%3Ahttps%3A%2F%2Fdatasheets.raspberrypi.com%2F%29%20There%20are>

[Accessed 23 Dec. 2022].

7. www.sparkfun.com. (n.d.). *SparkFun Environmental Sensor - BME688 (Qwiic) - SEN-19096 - SparkFun Electronics*. [online] Available at: <https://www.sparkfun.com/products/19096>.

8. learn.sparkfun.com. (n.d.). *SparkFun Environmental Sensor Breakout - BME68x (Qwiic) Hookup Guide - SparkFun Learn*. [online] Available at: <https://learn.sparkfun.com/tutorials/sparkfun-environmental-sensor-breakout---bme68x-qwiic-hookup-guide/all>

[Accessed 23 Dec. 2022].

9. community.bosch-sensortec.com. (2021). *BME688 and ESP32 driver compatibility*. [online] Available at: <https://community.bosch-sensortec.com/t5/MEMS-sensors-forum/BME688-and-ESP32-driver-compatibility/td-p/23969>

[Accessed 24 Dec. 2022].

10. Engineers Garage. (2019). *What is Breadboard?* [online] Available at: <https://www.engineersgarage.com/what-is-breadboard/>.

11. Wiltronics. (2022). *What Are Jumper Wires: Know by Colour, Types and Uses*. [online] Available at: <https://www.wiltronics.com.au/wiltronics-knowledge-base/what-are-jumper-wires/>.

12. Hemmings, M. (2018). *What is a Jumper Wire?* [online] blog.sparkfuneducation.com. Available at: <https://blog.sparkfuneducation.com/what-is-jumper-wire>.

13. www.paessler.com. (n.d.). *What is MQTT? Definition and Details*. [online] Available at: <https://www.paessler.com/it-explained/mqtt>.

14. www.catchpoint.com. (n.d.). *The Complete MQTT Broker Selection Guide*. [online] Available at:



<https://www.catchpoint.com/network-admin-guide/mqtt-broker>.

15. Eclipse Mosquitto. (2018). *Eclipse Mosquitto*. [online] Available at: <https://mosquitto.org/>.

16. www.catchpoint.com. (n.d.). *The Complete MQTT Broker Selection Guide*. [online] Available at: <https://www.catchpoint.com/network-admin-guide/mqtt-broker>.

17. OpenJS Foundation (2019). *Node-RED*. [online] Nodered.org. Available at: <https://nodered.org/>.

18. Noderedguide.com. (2010). *Node RED Programming Guide – Programming the IoT*. [online] Available at: <http://noderedguide.com/>.

19. Softonic. (n.d.). *PuTTY*. [online] Available at: <https://putty.en.softonic.com/>.