



Remote Monitoring of Coke Cans in a Fridge **using ESP32 and MQTT Protocol, with Real-time** **Updates on Smartphones**

Pursing the
Degree of Electronic and Telecommunication Engineering
Department of Engineering
SLTC Research University

Supervised by
Miss Vishaka Basnayake

Group members: M.T Ahmed -AA1505
Vishwa Lowe -AA1394
Geethma Jayasinghe -AA1442
Shanitha Alochana – AA1409
Vinod Yasintha-AA1495

ABSTRACT

This project focuses on the development of a temperature monitoring system for Coke cans in a fridge, providing real-time updates to a smartphone using the MQTT (Message Queuing Telemetry Transport) protocol. The system utilizes an ESP32 development board, a temperature sensor (such as DS18B20 or dht22), and an ultrasonic or IR sensor. By connecting these components to the ESP32 board, temperature readings are obtained and published on the "esp/temperature" topic every 5 seconds. Additionally, notifications are published on the "esp/notification" topic every 30 minutes, indicating the availability of Coke and the fridge temperature being below a predefined threshold. The MQTT broker, implemented on a Mosquitto Server, serves as the intermediary for communication between the ESP32 and a mobile app installed on a smartphone. Through the mobile app, users can visualize temperature readings in the form of a graph and receive text message notifications regarding the status of Coke bottles in the fridge. This system offers a convenient and efficient way to monitor and manage the temperature of beverages in a fridge remotely.

List of Contents

ABSTRACT.....	2
Introduction.....	5
What is Internet of things.....	5
Overview of project.....	5
Problem statement	5
Aim	5
Contribution	6
Literature Survey.....	7
Sensors	7
Hardware boards	8
MQTT protocol.....	8
MQTT client.....	9
MQTT broker.....	9
MQTT connection.....	9
IoT mobile applications	10
Methodology	11
Interfacing dht22 and Ultrasonic sensor with ESP32 Circuit Diagram	11
.....	11
Algorithm of the overall system.....	12
Specifications.....	13
Code	13
.....	17
Procedure of setting up the MQTT broker.....	19
Results and Discussion.....	21
Conclusion	23
References	24
Appendix.....	25
Appendix 1	25
Appendix 2.....	26

List of Figures

Figure 1 dht22 and Ultrasonic sensor with ESP32 Circuit Diagram	11
Figure 2 Flowchart of the overall system	12
Figure 3 Procedure of connect MQTT broker to ESP32	19
Figure 4 Mobile application	20
Figure 5 Result 1	21
Figure 6 Result 2	21
Figure 7 Result 3	22
Figure 8 Result 4	22

Introduction

What is Internet of things

The Internet of Things is referred to as IoT. In order for them to gather and exchange data, a network of interconnected physical items, including cars, buildings, and other objects, must be equipped with sensors, software, and network connectivity. These gadgets frequently include sensors and actuators built in so they can communicate with the real environment. IoT devices collect data, which may be evaluated and used to improve efficiency in a variety of industries, including manufacturing, transportation, healthcare, agriculture, and smart homes. A massive ecosystem of interconnected systems and devices is produced by the connectivity offered by the IoT. This connectivity allows objects to communicate with one another and with the internet.

Overview of project

This project involves developing a system that effectively monitors the temperature of Coke cans in a fridge and provides real-time updates to a smartphone. The system should implement the MQTT protocol in a ESP32 development board to publish temperature readings periodically and send notifications to the smartphone when certain conditions are met, such as the availability of Coke in the fridge and the temperature being below to a specified threshold. Additionally, the project aims to provide a user-friendly mobile app interface for viewing temperature graphs and receiving timely notifications about the status of the Coke bottles in the fridge.

Problem statement

The lack of efficient management for temperature control and availability of coke cans in the fridge requires the development of an automated system that can monitor temperature, ensure optimal temperature control, and provide real-time updates on coke can availability.

Aim

The project aims to develop a system that monitors the temperature of Coke cans in a fridge and offers timely updates to a smartphone using MQTT protocol, and sends notifications when Coke is available and the temperature is below a specified threshold.

Contribution

1. **Smart Monitoring:** The project demonstrates the application of IoT technology in monitoring the temperature of Coke cans in a fridge, showcasing the potential of IoT for real-time monitoring and control of various parameters in different environments.
2. **Efficient Data Communication:** By utilizing the MQTT protocol, the project establishes an efficient and lightweight communication mechanism between the ESP32 board and the smartphone app, showcasing the effectiveness of MQTT for IoT applications.
3. **Real-time Updates and Notifications:** The system provides real-time temperature updates to the smartphone, enabling users to monitor the temperature of the Coke cans remotely. Additionally, it sends notifications when specific conditions are met, enhancing user convenience and enabling timely actions.
4. **Integration of Sensors and Mobile App:** The project demonstrates the integration of sensors, such as temperature sensors, with a mobile app interface, highlighting the potential of IoT for integrating hardware devices and mobile applications to create comprehensive and user-friendly solutions.
5. **Data Visualization:** The mobile app interface allows users to view temperature readings as a graph over time, providing data visualization capabilities that enhance data interpretation and decision-making.

Literature Survey

Sensors

DHT 22 Sensor -



The DHT22 is a low-cost digital temperature and humidity sensor with a single wire digital interface. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed).

The sensor is calibrated and doesn't require extra components so you can get the right to measuring relative humidity and temperature.

Ultra Sonic Sensor -

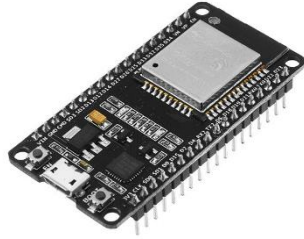


An ultrasonic sensor is a device that uses ultrasonic sound waves to measure distance or detect objects. It consists of a transmitter and a receiver. The transmitter emits high-frequency sound waves, usually above the range of human hearing, and the receiver detects the waves after they bounce off objects in the sensor's vicinity.

The time it takes for the sound waves to travel from the transmitter, hit an object, and return to the receiver is measured by the sensor. Using this time information and the speed of sound, the distance to the object can be calculated.

Hardware boards

The ESP32 development board



The ESP32 development board is widely used in the field of Internet of Things (IoT). It is based on the ESP32 microcontroller, which combines a various capability in a single chip.

- Dual-core processor - For efficient multitasking and handling complex applications.
- The built-in Wi-Fi and Bluetooth capabilities - Enable seamless wireless communication and connectivity.
- The abundance of GPIO pins - Allows for easy integration with a variety of sensors and peripheral devices.
- Memory resources - There is a generous amount of storage capacity for storing data and programs.
- Low power consumption capabilities - Make it suitable for battery-powered applications.
- Multiple languages - The board supports more than one programming languages and has an active community providing extensive support and resources.

MQTT protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency or unreliable networks. It is commonly used in Internet of Things (IoT) and machine-to-machine (M2M) communication scenarios.

It follows a publish/subscribe model where clients can publish messages to a broker, which then delivers them to subscribed clients. The protocol supports different levels of Quality of Service (QoS) to ensure message delivery reliability.

MQTT is highly efficient, making it suitable for resource-constrained devices and networks, and it also provides security features such as TLS encryption for secure communication.

How MQTT works,

1. An MQTT client establishes a connection with the MQTT broker.
2. Once connected, the client can either publish messages, subscribe to specific messages, or do both.
3. When the MQTT broker receives a message, it forwards it to subscribers who are interested.

MQTT client

An MQTT client is any device from a server to a microcontroller that runs an MQTT library. If the client is sending messages, it acts as a publisher, and if it is receiving messages, it acts as a receiver. Basically, any device that communicates using MQTT over a network can be called an MQTT client device.

MQTT broker

The MQTT broker is the backend system which coordinates messages between the different clients. Responsibilities of the broker include receiving and filtering messages, identifying clients subscribed to each message, and sending them the messages. It is also responsible for other tasks such as:

- Authorizing and authenticating MQTT clients
- Passing messages to other systems for further analysis
- Handling missed messages and client sessions

MQTT connection

Clients and brokers begin communicating by using an MQTT connection. Clients initiate the connection by sending a *CONNECT* message to the MQTT broker. The broker confirms that a connection has been established by responding with a *CONNACK* message. Both the MQTT client and the broker require a TCP/IP stack to communicate. Clients never connect with each other, only with the broker.

IoT mobile applications

Using a IoT mobile application typically allows the users to remotely control their IoT devices with real-time monitoring of the current status, sensor data, or other relevant information from their connected devices.

For our project we used,

REMOTE RED MOBILE APP



Remote-RED gives you mobile access to your Node-RED dashboard in the Laptop . It creates a tunnel between your computer network and your mobile device.

- This mobile app will serves as the user interface for monitoring the Coke cans' temperature and availability.
- Implement MQTT client functionality in the mobile app to subscribe to the MQTT topics used by the system.
- Receive and process MQTT messages from the broker in real-time to update the app's user interface.
- Display the current temperature readings on the app, along with the availability status of the Coke cans.

Methodology

The methodology for this project involves setting up the necessary hardware, establishing MQTT communication, and reading sensor data. The hardware setup includes connecting the DHT22 temperature and humidity sensor and configuring the Wi-Fi connection. MQTT communication is established by connecting to the MQTT broker, generating a unique client ID, and defining topics for publishing the sensor data. Finally, the sensor data is read from the DHT22 sensor, including temperature and humidity values. These values are then published to the MQTT broker using the defined topics.

Interfacing dht22 and Ultrasonic sensor with ESP32 Circuit Diagram

Here is the set up structure we have utilized. There are four pins (which are trigger, echo, Vcc and Vin) in ultrasonic sensor which are connected to esp32. And also DHT22 Sensor has been connected to esp32 using relevant pins according to the given instructions.

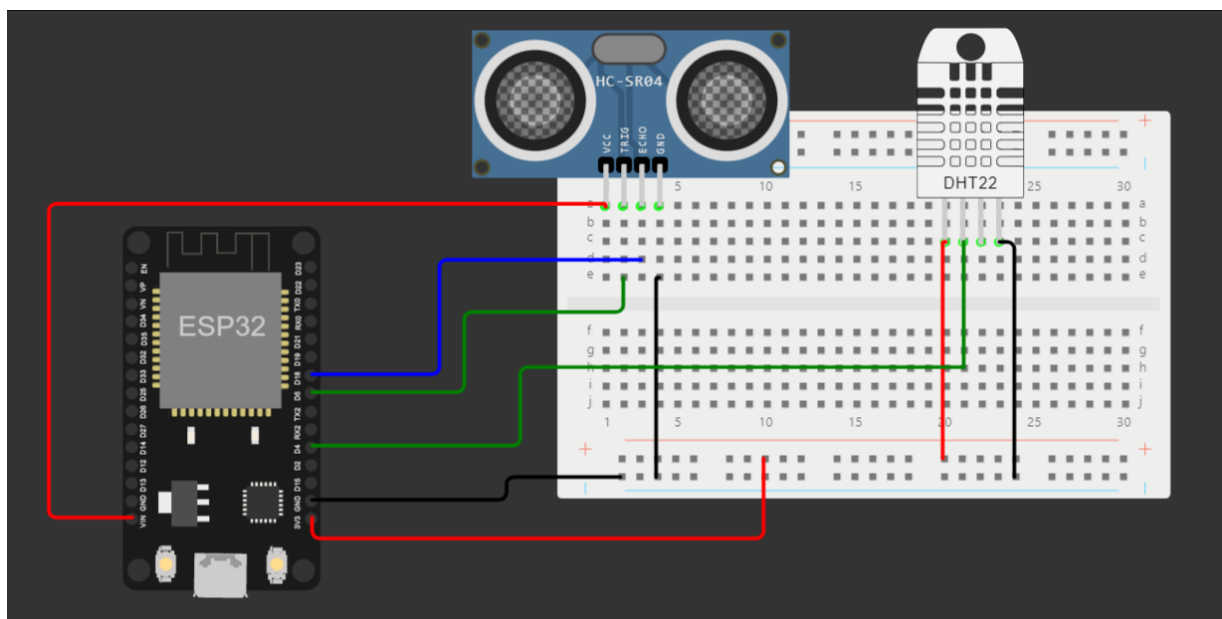


Figure 1 dht22 and Ultrasonic sensor with ESP32 Circuit Diagram

Algorithm of the overall system

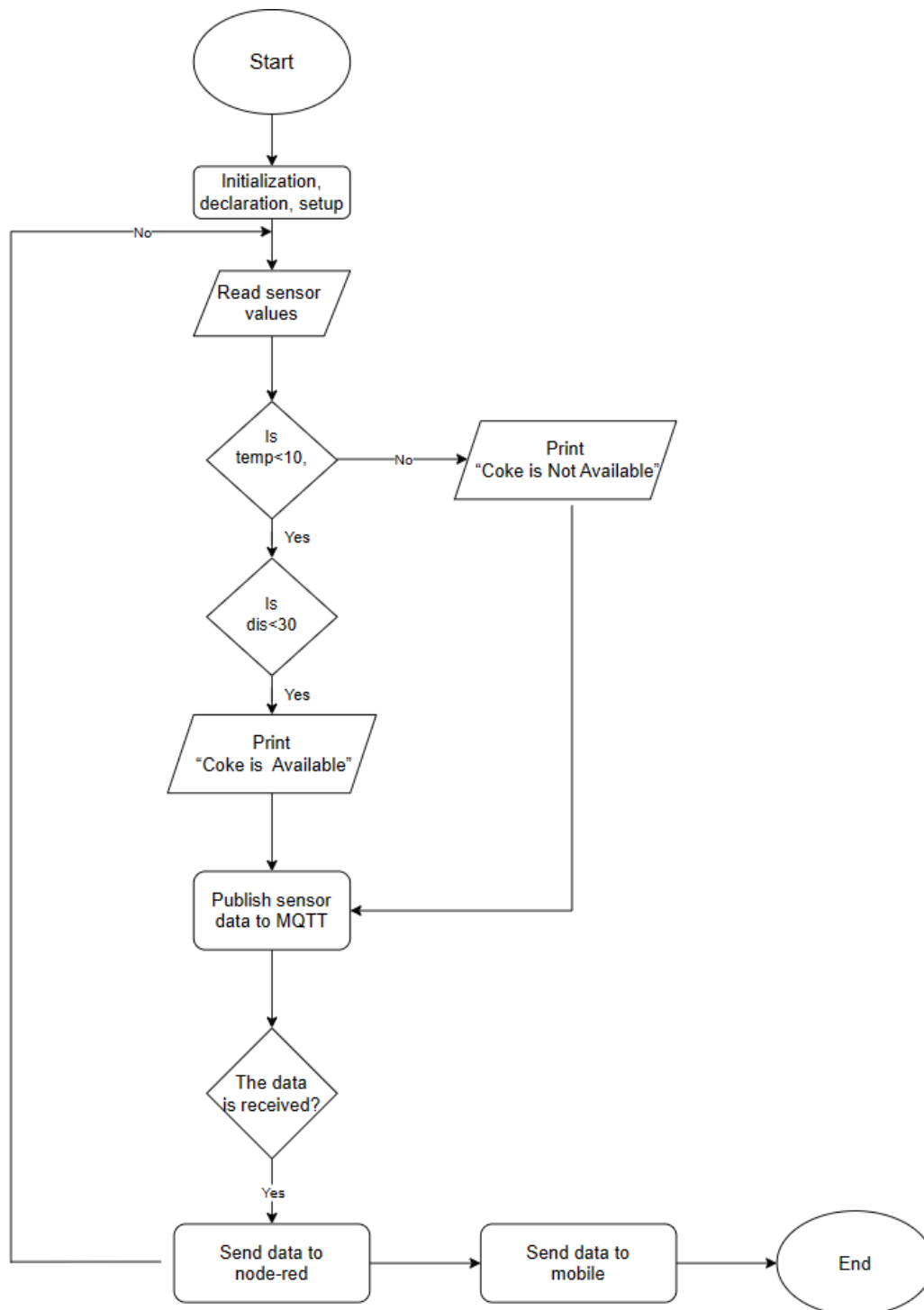


Figure 2 Flowchart of the overall system

Specifications

		Specifications
Hardware	ESP32 Development Board	Microcontroller with Wi-Fi connectivity and GPIO pins
	Temperature Sensor	DS18B20 or DHT11 (or similar) temperature sensor
	Ultrasonic Sensor	Sensor for detecting the presence of Coke cans in the fridge
	Jumper Wires	Wiring connections between the ESP32 board and the sensors
	Breadboard	Platform for temporary connections during prototyping
	USB Cable	Connects the ESP32 board to the computer for programming and power
	Coke Cans and Fridge	Actual Coke cans and a fridge for simulating real-world scenario
Software	Micropython	Lightweight implementation of Python for ESP32 programming
	Thonny IDE	Integrated development environment (IDE) for Micropython
	MQTT Protocol	Message Queuing Telemetry Transport protocol for communication
	Mosquitto Server	Open-source MQTT broker for message distribution and reception
	Mobile App	Smartphone application for receiving temperature readings and notifications

Code

1)

In this section, the necessary modules and libraries are imported. Here's a brief explanation of each import:

- **import time:** This module provides functions for working with time-related operations, such as delays.
- **from umqttsimple import MQTTClient:** The **MQTTClient** class from the **umqttsimple** module is imported. It provides a simple MQTT client implementation.
- **import ubinascii:** This module provides functions for working with binary and ASCII data, such as converting binary data to hexadecimal representation.
- **import machine:** The **machine** module provides access to hardware-specific functions and features, such as controlling GPIO pins.

- **import micropython:** This module allows interaction with the MicroPython runtime, providing functionalities like garbage collection control.
- **import network:** The **network** module provides functionality for connecting to Wi-Fi networks.
- **import esp:** The **esp** module provides functions related to the ESP8266/ESP32 SoC (System on Chip), such as debugging options.
- **from machine import Pin:** The **Pin** class from the **machine** module is imported. It allows control over GPIO pins.
- **import dht:** This module provides functions for working with DHT temperature and humidity sensors.
- **esp.osdebug(None):** This line disables the debugging output of the ESP8266/ESP32 SoC.
- **import gc:** The **gc** module provides functions for garbage collection in MicroPython.

```
main7.py ×
1  import time
2  from umqttsimple import MQTTClient
3  from hcsr04 import HCSR04
4  from time import sleep
5  import ubinascii
6  import machine
7  import micropython
8  import network
9  import esp
10 from machine import Pin
11 import dht
12
13 esp.osdebug(None)
14 import gc
15 gc.collect()
```

2) In this section, define the network and MQTT settings:

- **ssid**: Replace '**REPLACE_WITH_YOUR_SSID**' with the SSID (name) of your Wi-Fi network.
- **password**: Replace '**REPLACE_WITH_YOUR_PASSWORD**' with the password for your Wi-Fi network.
- **mqtt_server**: Replace '**192.168.1.XXX**' with the IP address or domain name of your MQTT broker server.
- **client_id**: A unique identifier for the MQTT client. It is generated using the **unique_id()** method of the **machine** module and converted to hexadecimal using **ubinascii.hexlify()**.
- **topic_pub_temp**: The topic to which the temperature data will be published. It is set as **b'esp/dht/temperature'**, where **b** indicates that it is a byte string.
- **topic_pub_dis**: The topic to which the humidity data will be published. It is set as **b'esp/dht/distance'**, also a byte string.

```
16
17 ssid = 'DESKTOP-I2S0M9H 9142'
18 password = '12345678'
19 mqtt_server = 'test.mosquitto.org'
20
21 client_id = ubinascii.hexlify(machine.unique_id())
22
23 topic_pub_temp = b'esp/dht/temperature'
24 topic_pub_dis = b'esp/dht/distance'
25
```

3) Here, two variables are initialized:

- **last_message**: A timestamp representing the last time a message was published. It is initially set to 0.
- **message_interval**: The time interval (in seconds) between publishing consecutive messages. It is set to 5 seconds.

```
26 last_message = 0
27 message_interval = 5
28 counter = 0
29
```

4) This section sets up a Wi-Fi connection to the specified network:

- **station**: An instance of the **WLAN** class from the **network** module, which represents the Wi-Fi station interface.
- **station.active(True)**: Enables the Wi-Fi station interface.
- **station.connect(ssid, password)**: Attempts to connect to the specified Wi-Fi network using the provided SSID and password.
- The **while** loop waits until the station is successfully connected to the network (**station.isconnected() == True**).
- Once connected, the message "Connection successful" is printed

```
29
30 station = network.WLAN(network.STA_IF)
31 station.active(True)
32 station.connect(ssid, password)
33
34 while not station.isconnected():
35     pass
36
37 print('Connection successful')
38
```

5) This section initializes the sensors:

```
38
39 sensor = dht.DHT22(Pin(14))
40 sensor1 = HCSR04(trigger_pin=5, echo_pin=18, echo_timeout_us=10000)
41
```

6) This section defines a function **connect_mqtt()** that establishes a connection to the MQTT broker:

- The function creates an instance of the **MQTTClient** class, passing the **client_id** and **mqtt_server** as arguments.
- The **client.connect()** method is called to establish the connection to the MQTT broker.
- Once connected, the message "Connected to MQTT broker" is printed, and the client object is returned.

```
42
43 def connect_mqtt():
44     global client_id, mqtt_server
45     client = MQTTClient(client_id, mqtt_server)
46     client.connect()
47     print('Connected to %s MQTT broker' % mqtt_server)
48     return client
49
50
```


7) This section defines a function `restart_and_reconnect()` to handle reconnection to the MQTT broker in case of failure:

```
50
51 def restart_and_reconnect():
52     print('Failed to connect to MQTT broker. Reconnecting...')
53     time.sleep(10)
54     machine.reset()
```

8) This section defines a function `read_sensor()` to read temperature and humidity data from the DHT22 sensor:

- The function calls `sensor.measure()` to take a measurement from the sensor.
- The temperature and humidity values are obtained using `sensor.temperature()` and `sensor.humidity()` methods, respectively.
- ```
56
57 def read_sensor():
58 try:
59 sensor.measure()
60 temp = int(sensor.temperature())
61 hum = int(sensor.humidity())
62 if isinstance(temp, (int, float)) and isinstance(hum, (int, float)):
63 return temp
64 else:
65 return None
66 except OSError as e:
67 return None
68
```

9) This section attempts to establish an MQTT connection by calling the `connect_mqtt()` function. If an `OSError` occurs (indicating a failure to connect), it calls the `restart_and_reconnect()` function to handle the reconnection process.

```
69
70 try:
71 client = connect_mqtt()
72 except OSError as e:
73 restart_and_reconnect()
74
```

10) This section contains an infinite loop that continuously reads sensor data and publishes it to the MQTT broker.

```
75 while True:
76 try:
77 if (time.time() - last_message) > message_interval:
78 temp = read_sensor()
79 distance = int(sensor1.distance_cm())
80 print(temp)
81 print(distance)
82 if temp is not None and distance is not None:
83 if temp < 40 and distance < 30:
84 availability = b'Coke is available'
85 else:
86 availability = b'Coke is not available'
87
88 print(availability)
89
90 client.publish(b'esp/coke/availability', availability)
91 client.publish(topic_pub_temp, str(temp).encode())
92 client.publish(topic_pub_dis, str(distance).encode())
93 last_message = time.time()
94 counter += 1
95 except OSError as e:
96 restart_and_reconnect()
97
```

If an **OSError** occurs during this process, the **restart\_and\_reconnect()** function is called to handle the reconnection to the MQTT broker.

## Procedure of setting up the MQTT broker.

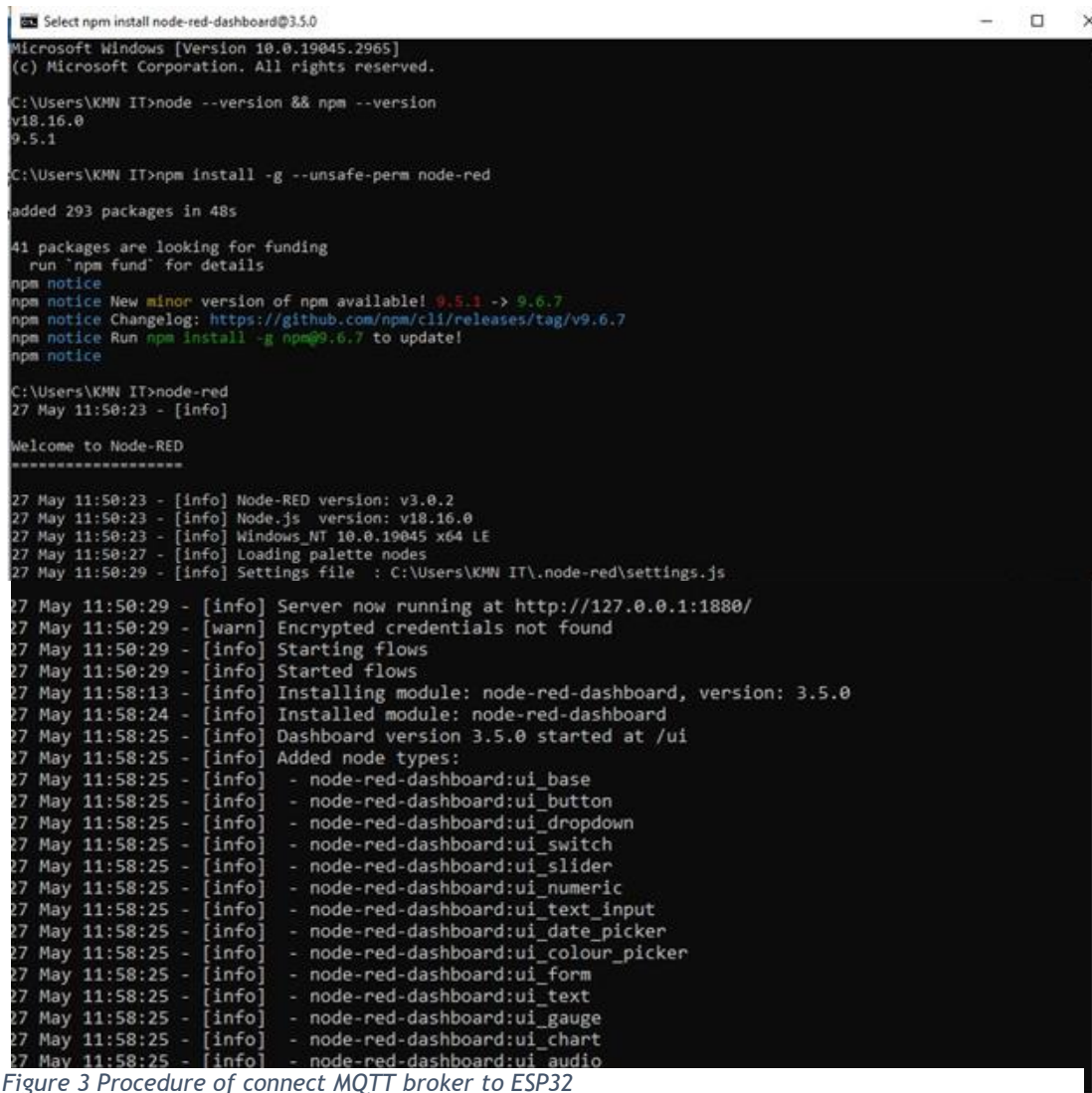
Here is the procedure that how do we connect mqtt broker to esp32 which we used in our whole project. We have used mosquitto protocol that we have instructed.so you are able to visible the steps we followed .

Install the Mosquitto broker on a server or use a cloud-based MQTT broker service like CloudMQTT or HiveMQ.

Configure the MQTT broker with the desired settings such as port number, authentication, and topic structure.

In your ESP32 project, include the required libraries and define the necessary variables for connecting to the MQTT broker. Set up the Wi-Fi connection on the ESP32 to connect to your local network.

Set up the MQTT client by providing the broker's **IP address or hostname**, **port number**, and **any required authentication details**. Define the callback functions for handling incoming messages and managing the MQTT connection state. Connect to the MQTT broker by calling the appropriate function in your code. Subscribe to specific MQTT topics or publish messages to topics as needed.



```
Select npm install node-red-dashboard@3.5.0
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\KMN IT>node --version && npm --version
v18.16.0
9.5.1

C:\Users\KMN IT>npm install -g --unsafe-perm node-red

added 293 packages in 48s

41 packages are looking for funding
 run 'npm fund' for details

npm notice
npm notice New minor version of npm available! 9.5.1 -> 9.6.7
npm notice Changelog: https://github.com/npm/cli/releases/tag/v9.6.7
npm notice Run npm install -g npm@9.6.7 to update!
npm notice

C:\Users\KMN IT>node-red
27 May 11:50:23 - [info]
Welcome to Node-RED

27 May 11:50:23 - [info] Node-RED version: v3.0.2
27 May 11:50:23 - [info] Node.js version: v18.16.0
27 May 11:50:23 - [info] Windows_NT 10.0.19045 x64 LE
27 May 11:50:27 - [info] Loading palette nodes
27 May 11:50:29 - [info] Settings file : C:\Users\KMN IT\.node-red\settings.js

27 May 11:50:29 - [info] Server now running at http://127.0.0.1:1880/
27 May 11:50:29 - [warn] Encrypted credentials not found
27 May 11:50:29 - [info] Starting flows
27 May 11:50:29 - [info] Started flows
27 May 11:58:13 - [info] Installing module: node-red-dashboard, version: 3.5.0
27 May 11:58:24 - [info] Installed module: node-red-dashboard
27 May 11:58:25 - [info] Dashboard version 3.5.0 started at /ui
27 May 11:58:25 - [info] Added node types:
27 May 11:58:25 - [info] - node-red-dashboard:ui_base
27 May 11:58:25 - [info] - node-red-dashboard:ui_button
27 May 11:58:25 - [info] - node-red-dashboard:ui_dropdown
27 May 11:58:25 - [info] - node-red-dashboard:ui_switch
27 May 11:58:25 - [info] - node-red-dashboard:ui_slider
27 May 11:58:25 - [info] - node-red-dashboard:ui_numeric
27 May 11:58:25 - [info] - node-red-dashboard:ui_text_input
27 May 11:58:25 - [info] - node-red-dashboard:ui_date_picker
27 May 11:58:25 - [info] - node-red-dashboard:ui_colour_picker
27 May 11:58:25 - [info] - node-red-dashboard:ui_form
27 May 11:58:25 - [info] - node-red-dashboard:ui_text
27 May 11:58:25 - [info] - node-red-dashboard:ui_gauge
27 May 11:58:25 - [info] - node-red-dashboard:ui_chart
27 May 11:58:25 - [info] - node-red-dashboard:ui_audio
```

Figure 3 Procedure of connect MQTT broker to ESP32

Method used in connecting ESP 32 and the smart phone to the MQTTbroker,

- We had a MQTT broker in place. we have used Mosquitto as cloud based mqtt broker.
- On the smartphone side, we have utilized develop mobile application which is remote red .



*Figure 4 Mobile application*

## Results and Discussion

Here we have established the necessary stuff that related to project. So initially you can see that the temperature was 10C. Which means The temperature was above the threshold value. So the output result has shown as **coke is not available**.

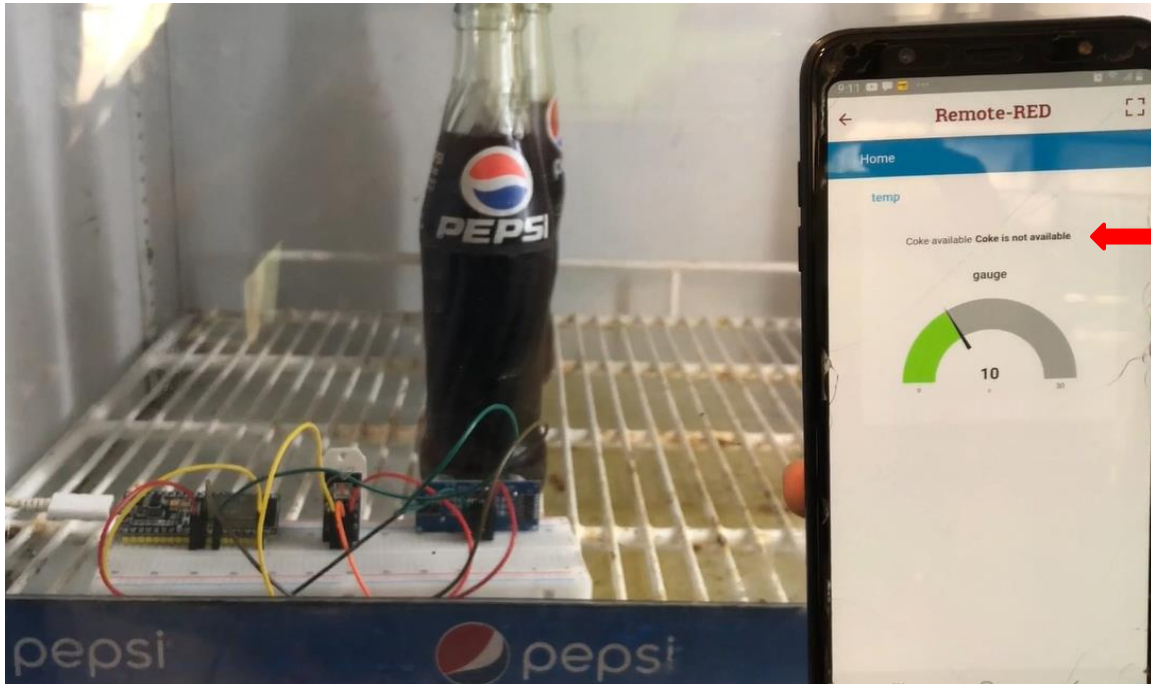


Figure 5 Result 1

Now the temperature has been deducted in to 9C. then the output is **coke is available** . and also we are supposed to figure out , the distance between coke bottles and the sensor is well less than 30cm.



Figure 6 Result 2



So here we were going to explore the availability of Coke bottles using the distance. At the movement we had been reducing the bottles one by one over there. Let's see what was going on?

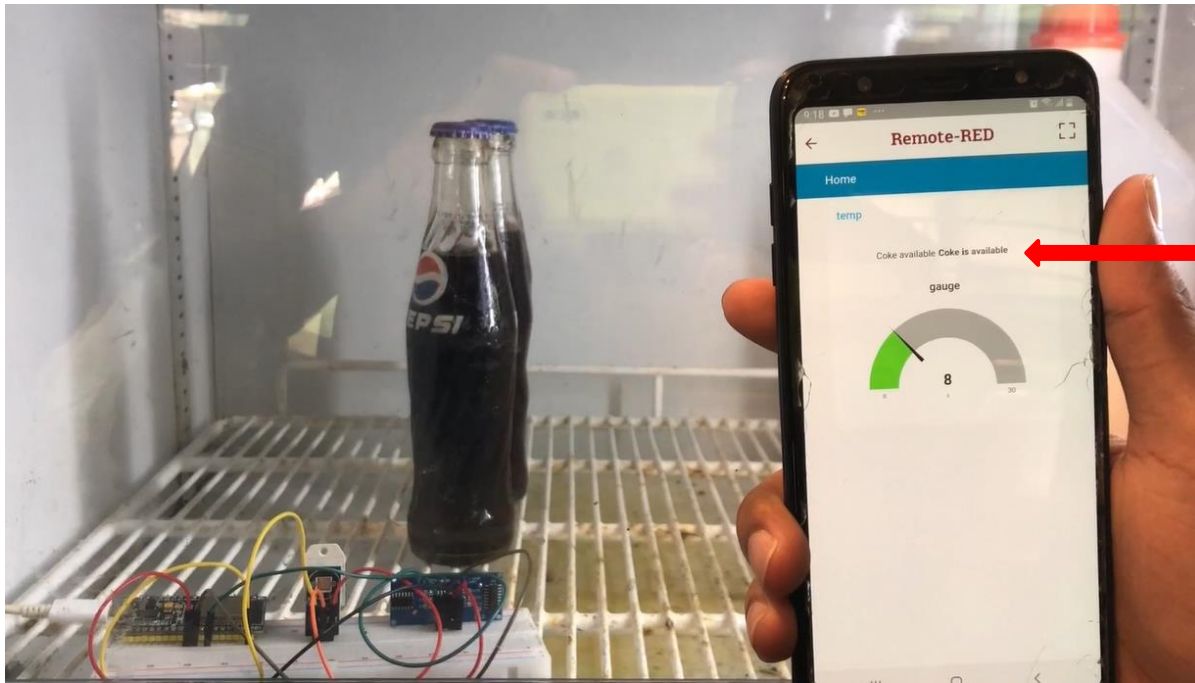


Figure 7 Result 3

In here we have removed all the bottled out there. And the temperature was below the threshold value too. However, the output was **coke is not available**. Which means that we can figure out there are no cokes. so here we are able to proof when the distance is over than 30cm, coke is not available.

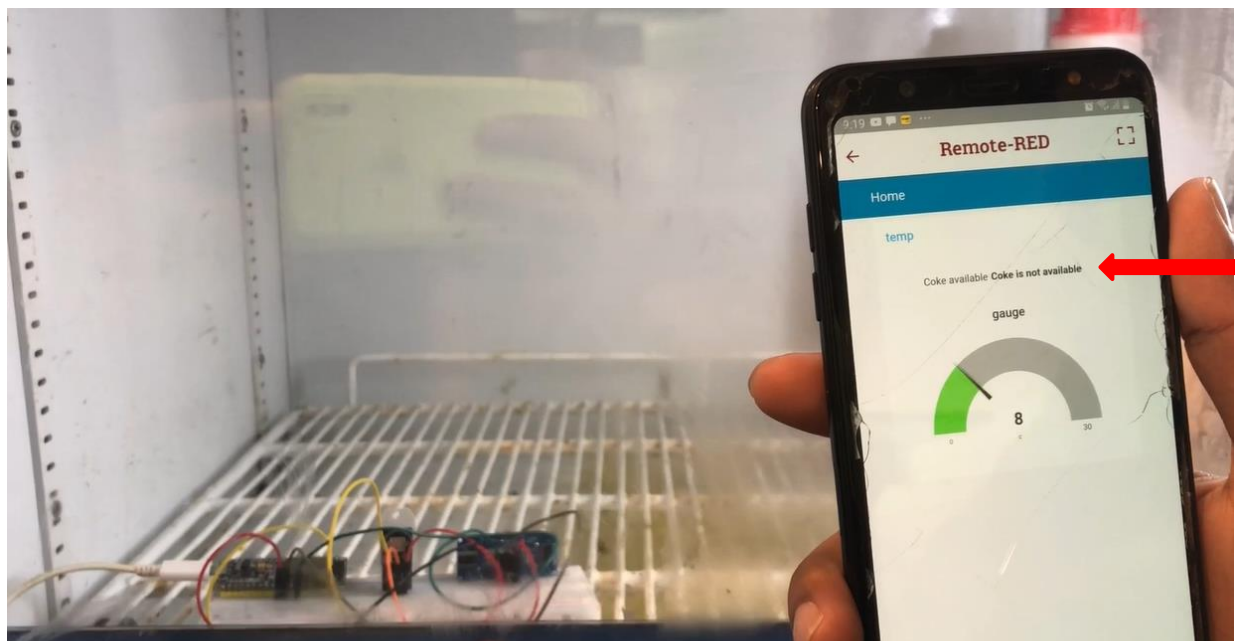


Figure 8 Result 4

# Conclusion

This project demonstrates how to monitor the temperature of Coke cans in a fridge and send real-time updates to a smartphone using the MQTT protocol. The project highlights the use of an ESP32 development board, temperature sensor, and an ultrasonic sensor. These components are connected and programmed to publish temperature readings to an MQTT broker at regular intervals. Additionally, the system sends a notification to the smartphone app when certain conditions are met, indicating the availability of Coke cans and a temperature below a specified threshold. By implementing the MQTT broker on a Mosquitto Server and connecting it to a mobile app, users can view temperature readings as a graph over time and receive text message notifications about the status of Coke bottles in the fridge.

Overall, this project serves as a foundation for building a more comprehensive temperature control and monitoring system, and it can be expanded upon to meet specific requirements or integrate with other systems for enhanced functionality and automation.

# References

- <https://aws.amazon.com/whatis/mqtt/#:~:text=MQTT%20is%20a%20standards%2Dbase,d.constrained%20network%20with%20limited%20bandwidth.>
- <https://apps.apple.com/us/app/remote-red/id1529777665>
- <https://cityos-air.readme.io/docs/4-dht22-digital-temperature-humidity-sensor>
- <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf>
- <https://www.electronicwings.com/sensors-modules/dht11>
- <https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm/related>
- <https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/>
- <https://maxbotix.com/blogs/blog/how-ultrasonic-sensors-work>
- <https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/>
- <https://www.youtube.com/watch?v=P8DnANWusrM>
- <https://www.youtube.com/watch?v=hyJhKWhxAxA>
-



# Appendix

## Appendix 1

### Code

```
import time
from umqttsimple import MQTTClient
from hcsr04 import HCSR04
from time import sleep
import ubinascii
import machine
import micropython
import network
import esp
from machine import Pin
import dht
esp.osdebug(None)
import gc
gc.collect()

def read_sensor():
 try:
 sensor.measure()
 temp = int(sensor.temperature())
 hum = int(sensor.humidity())
 if (isinstance(temp, float) and isinstance(hum, float)) or
 (isinstance(temp, int) and isinstance(hum, int)):
 #temp = (b'{0:3.1f}', '.format(temp))
 #hum = (b'{0:3.1f}', '.format(hum))

 # uncomment for Fahrenheit
 #temp = temp * (9/5) + 32.0
 return temp
 else:
 return('Invalid sensor readings.')
 except OSError as e:
 return

sensor = dht.DHT22(Pin(14))
sensor1 = HCSR04(trigger_pin=17, echo_pin=18, echo_timeout_us=10000)

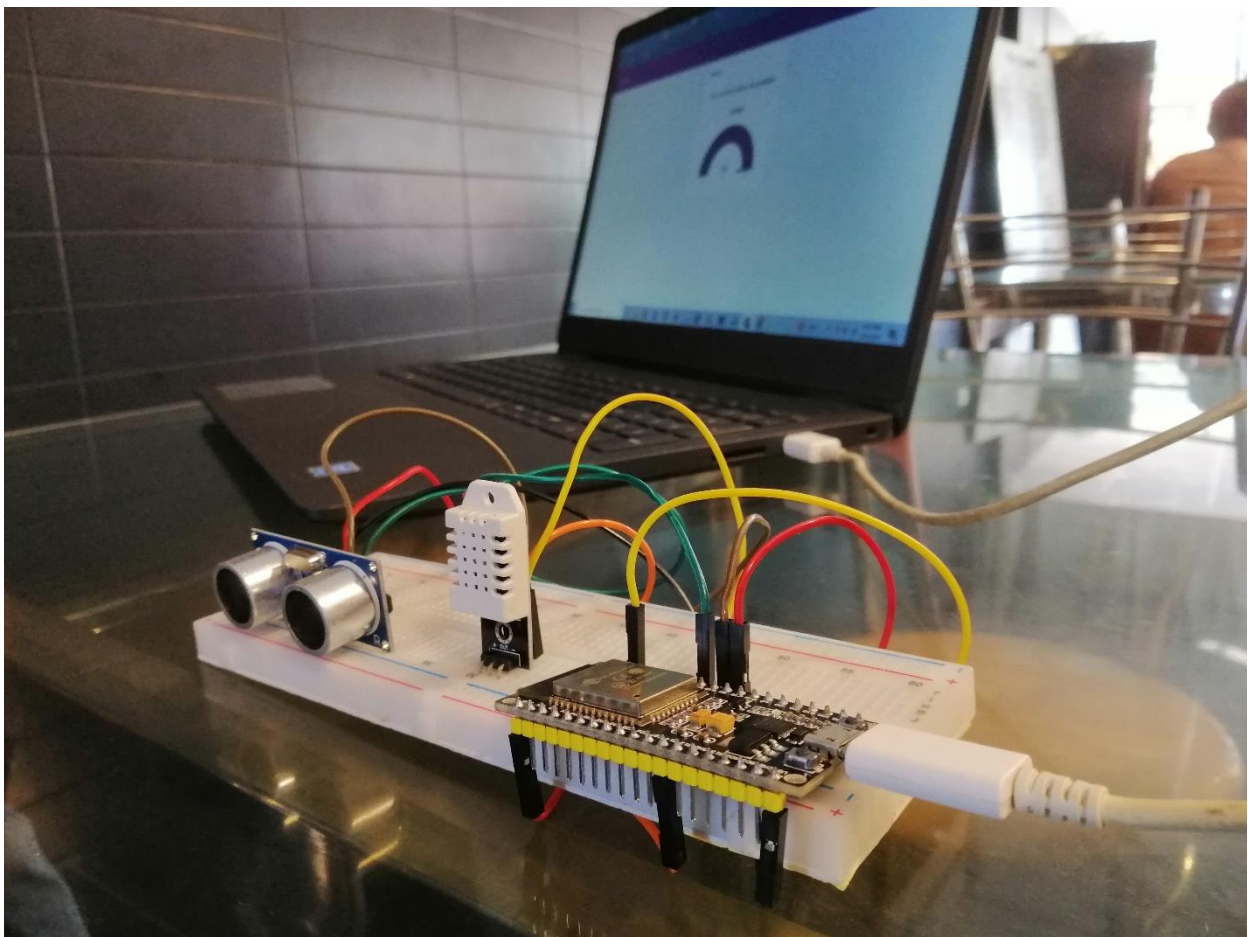
while True:

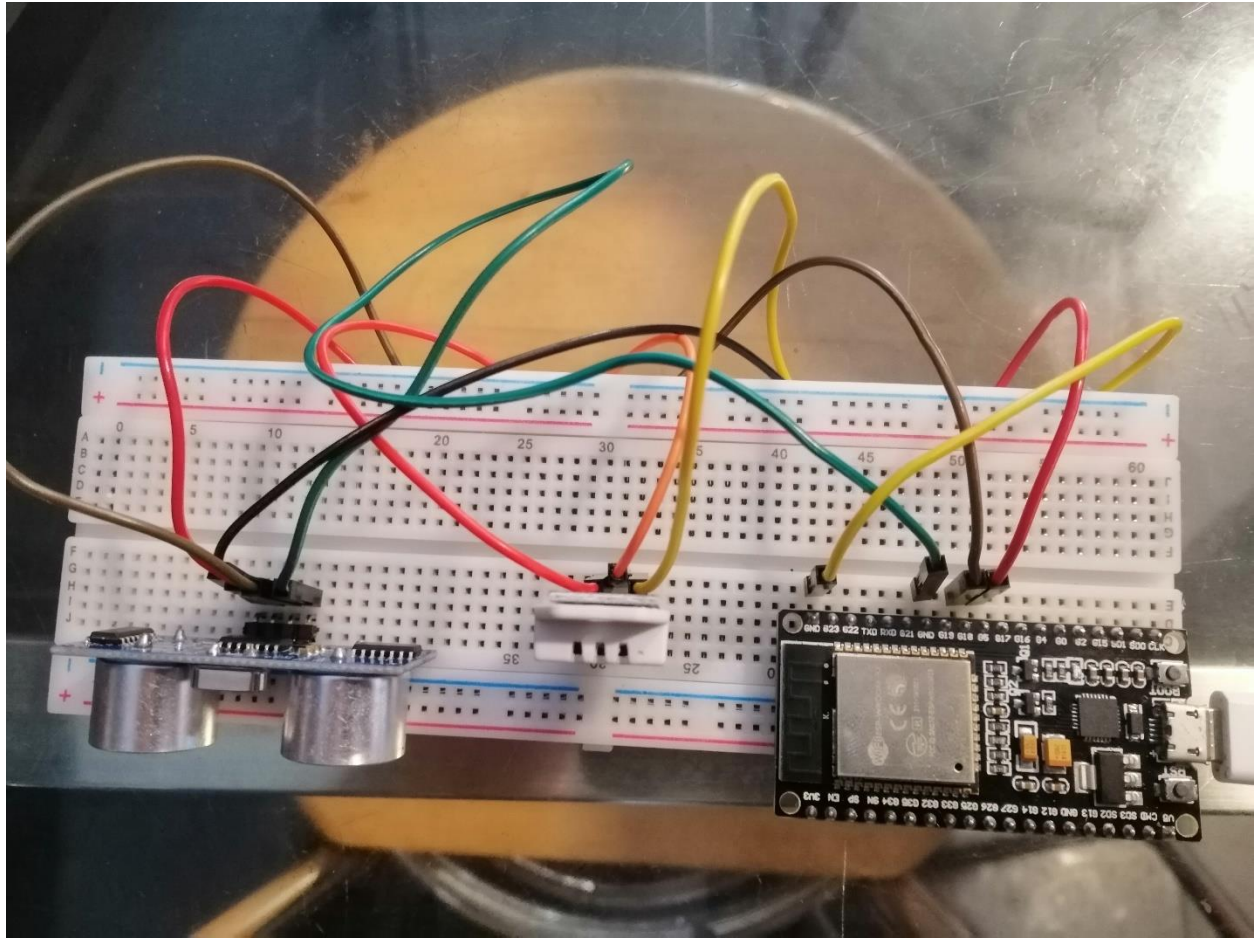
 temp1 = read_sensor()
 distance =int(sensor1.distance_cm())

 if temp1 != None:
 print('Distance:', distance, 'cm')
 print('Temperature:', temp1, 'C')
```

```
if temp1 < 8 and distance < 30:
 print('Coke Available')
else:
 print('Coke is not Available')
sleep(0.1)
```

## Appendix 2





**THE END!**