



School of Engineering and Applied Sciences
Electronics and Computer Engineering (ECE)

Wireless Communication

ECEN-463

Final Project Report

BY:

ALHassan Khaled	18100745
Abdullah Hammad	18102381
Adham Nassar	18101060
Adham Maher	19105072
Mazen khattab	18101368
Youssef Shawky	18102428

Submitted to:

Dr. Tawfik Ismael

Fall 2022

Contents

Project Idea	3
Methodology	4
Workflow	6
Results Discussion	8

Project Idea

Internet of Things is an interesting topic which empowers everyone to automate simple or everyday tasks. Our prototype basically detects motion on a room entrance and sense light, if motion detected in the absence of light the LED will get turned on, any other case such as motion detection with presence of light (for example: sunlight) won't trigger the LED. our project aims to automate light systems to help save electricity and reduce bills to support economy and reduce the human factor in such scenario. Also add more smart features such as remote monitoring and mobile control. Environment control system using Raspberry pi, NodeMcu (ESP8266) Microcontroller, DHT 11 Sensor (temperature & humidity), DC Motor (Fan) and LED. Sensor data is produced from the ESP8266 to Node-RED. The Node-RED is a software running on a Raspberry Pi, and the communication between the ESP8266 and the Node-RED software is achieved with the MQTT communication protocol via a broker (mosquito) installed on the raspberry pi.

Methodology

We used **NodeMCU** as the main controller connected to **light-dependent resistor** (LDR) and **Passive Infrared Sensor** (PIR). these sensors then send their readings to the controller and upon the right conditions LED gets triggered, also when the conditions fail to meet the required values the LED gets turned off. Once the readings reach the controller. It then publishes these readings through the raspberry pi controller which acts as a broker in a MQTT protocol. It then sends the readings through a certain topic which on the other end is subscribed to by NodeRED. NodeRED acts as a locally hosted dashboard which shows the sensor readings and control a backup LED for an overridden control in emergency situations. We also connected MQTT Dashboard mobile application with the locally hosted system so we can control the LED freely without hassle. Google Firebase was also induced to print our Realtime sensor readings. In this project we established a communication between a Raspberry Pi running the Node-RED software and an ESP8266 using MQTT. MQTT stands for MQ Telemetry Transport, and it is a lightweight publish & subscribe system where you can publish and receive messages as a client. It is a simple messaging protocol, designed for constrained devices and with low bandwidth. So, it's the perfect solution for Internet of Things applications.

In MQTT, the broker is primarily responsible for receiving all messages, filtering the messages, deciding where to send it and then publishing the message to all subscribed clients. There are several brokers that can be used. In this project we used the **Mosquitto Broker** which was installed on the Raspberry Pi.

Next, we created the Node-RED dashboard layout. We created a switch to control a led from ESP8266 output; a chart and a gauge to display temperature and humidity readings from the DHT11 temperature and humidity sensor. After that, we started creating our nodes, set our topics,

and programming the logic of our system on Node-RED. We used a firebase to be as a cloud server. As firebase can get the data (temperature and humidity) from NodeMcu. We also used a MQTT Dashboard – IoT and Nodes app and made a confirmation of all topics to work with Node-RED as a mobile application and can show Temperature, Humidity and also can switch the LED to turn ON or OFF.

Furthermore, we started connecting our NodeMcu circuit. The whole circuit essentially consisted of ESP8266, the DHT 11 Sensor, DC Motor (Fan), Relay, 12v Battery, 330Ω Resistors and LEDs.

Our topics which the sensor and actuators must subscribe to publish on it are:

Topic	Usage	
room/lamp	Turn ON or OFF the LED	Node-RED
room/lamp/button	Make the button of the app synchronized with Node-RED Platform	Node-RED
room/temperature	Show temperature	Node-RED
room/humidity	Show humidity	Node-RED
room/hightemperature	Alarm to send an Email if the temperature is above the safe level	Node-RED
DHT 11/temp	Show temperature	Firebase
DHT 11/hum	Show humidity	Firebase
room/Lux	Show light intensity	Node-RED
room/Pir	Show State	Node-RED

Workflow

Researched the idea and collected needed tools and studied the basics concepts of the required software. Built the circuit and tested every element to avoid faulty instruments. Built mosquito protocol and configured topics and subscriptions on each node. Built the dashboard on NodeRED for detailed readings view. Configured MQTT Dashboard mobile application to control the LED from mobile phone. Linked Firebase with NodeMCU so we can get Realtime sensor readings. Configured Adafruit to know if a motion is detected or not.

Stage 1

We started by collecting required components including but not limited to:

- SD card to boot raspberry pi Image on raspberry pi.
- Sensors
- Microcontroller (ESP8266)
- Connectors and batteries

Then we started installing required software components on the raspberry pi which included Node-RED and Mosquito. We also established the suitable type of connection between the raspberry pi and the PC which was SSH.

Stage 2

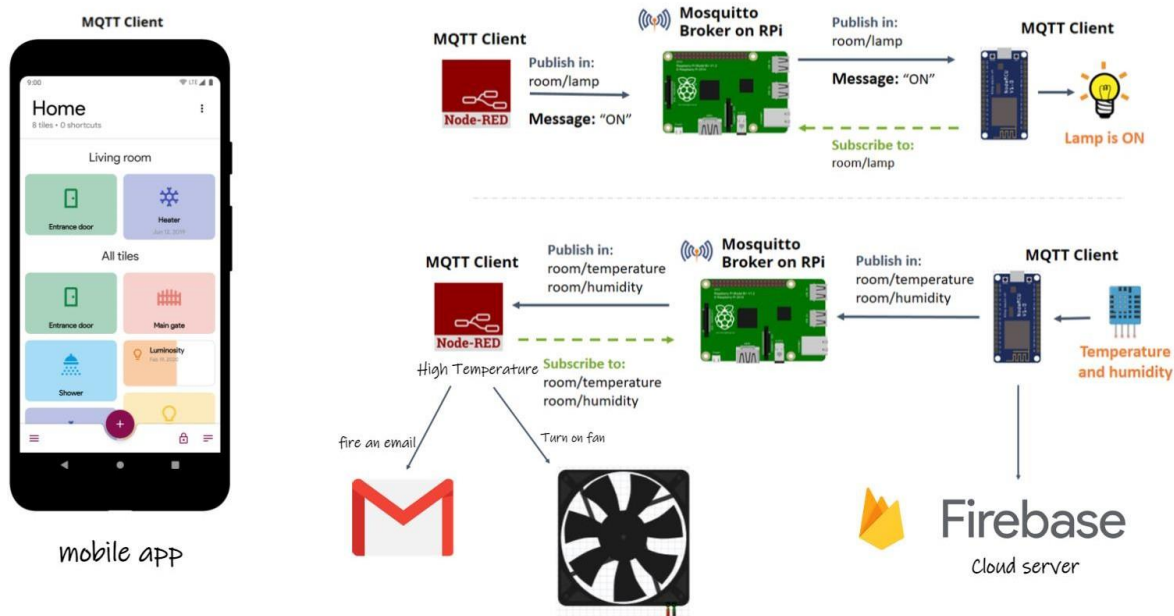
In this stage, everything we needed was ready and we started by connecting our circuit and testing it. Then we created the Node-RED dashboard and started implementing our flow-based programming to our dashboard. In this stage several aspects were tested, and several initial bugs and difficulties were resolved. Most importantly, RPi code was written and debugged While establishing Inter Process Communication. At the end of this stage, we were able to establish communication between the Raspberry Pi running the Node-RED software and an ESP8266 circuit using MQTT protocol. Our circuit was running properly.

Stage 3

In this stage we focused on implementing 3 features:

- Remote cloud access using Firebase.
- WLAN mobile access using MQTT dashboard app.
- E-mail notification system using Node-RED.

In this stage everything went according to plan with minimal setbacks. We were able to end up with aviable email system that sends a warning mail whenever temperature exceeds safe level. We were alsoable to access our dashboard either from a local network or remotely through the cloud. All required dashboards were properly implemented and tested.



Results Discussion

At last, we built a fully operational IoT system that can control and automate the light system in anyplace. We selected the MQTT protocol to enable multi-systems and guarantee instant responses, this project could be done by using raspberry pi only instead of using it with a NodeMCU, but we mixed both to allow many systems to be present with one broker, because Raspberry Pi is relatively costly compared to the NodeMCU. In a big house, we can allocate a NodeMCU to every room with its sensors and only use one Raspberry Pi to handle the communication for all of them. Thus, it won't be very expensive to implement this way. Also, MQTT has shown better and improved performance over simple http get requests and was easy to implement. That's why it was the convenient choice. You can have the system fully automated or override it using your phone with ease. But for prototyping reasons, you can only control this system from the local network only as it's locally hosted, however, you may follow the sensors reading through firebase dashboard or follow motion sense for security control from Adafruit.

For future work, this project can have more sensors and control more variables such as: temperature and humidity sensor to control room AC, flame sensor for fire alarm and much more.

This project can be considered as a Smart Home/Automation system.

Workflow Demonstration

Furthermore, we started connecting our NodeMcu circuit. The whole circuit essentially consisted of ESP8266, the DHT 11 Sensor, DC Motor (Fan), Relay, 12v Battery, 330 Ω Resistors and LEDs.

The schematic was as follows:

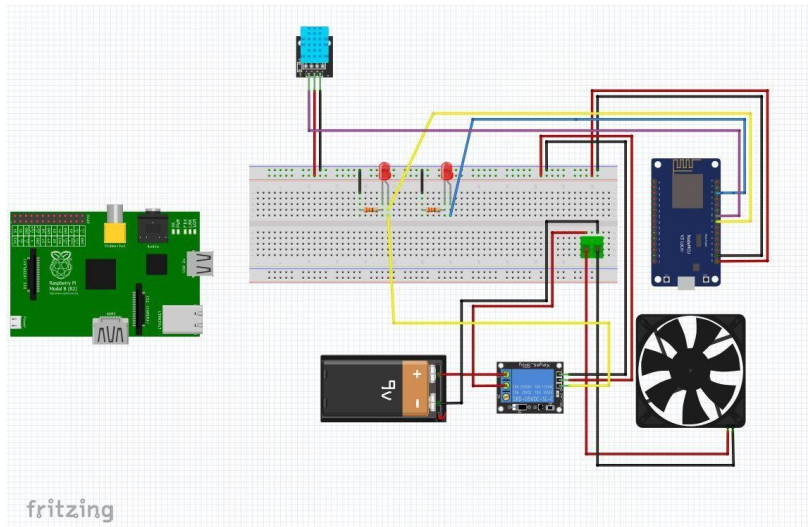


Figure 1. Workflow Block Diagram made on <https://app.diagrams.net>

Edge Server schematic circuit, we take value of lux from the output of the voltage divider circuit the LDR connected in it and the PIR works with no library with one output digital pin

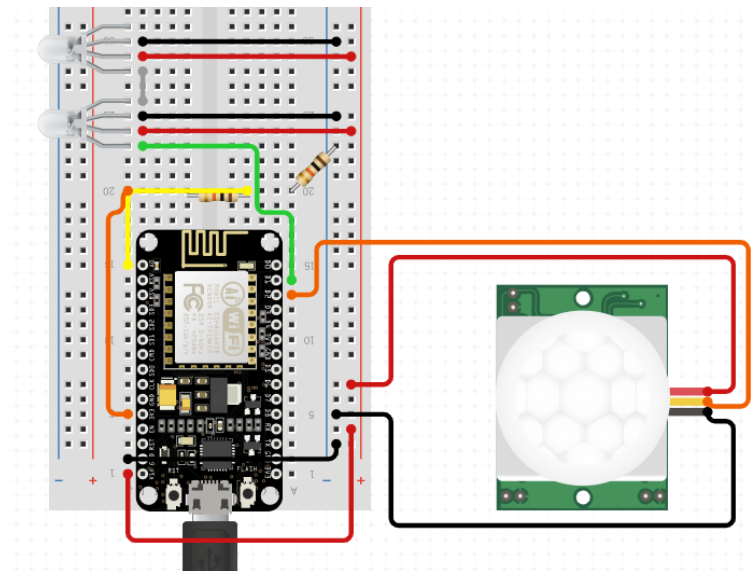


Figure 2.Edge Server

What is MQTT?

Message Queuing Telemetry Transport is referred to as MQTT. A straightforward communications protocol called MQTT was created for limited-bandwidth devices. Therefore, it's the ideal method for transferring data between various IoT devices.

MQTT – Publish/Subscribe

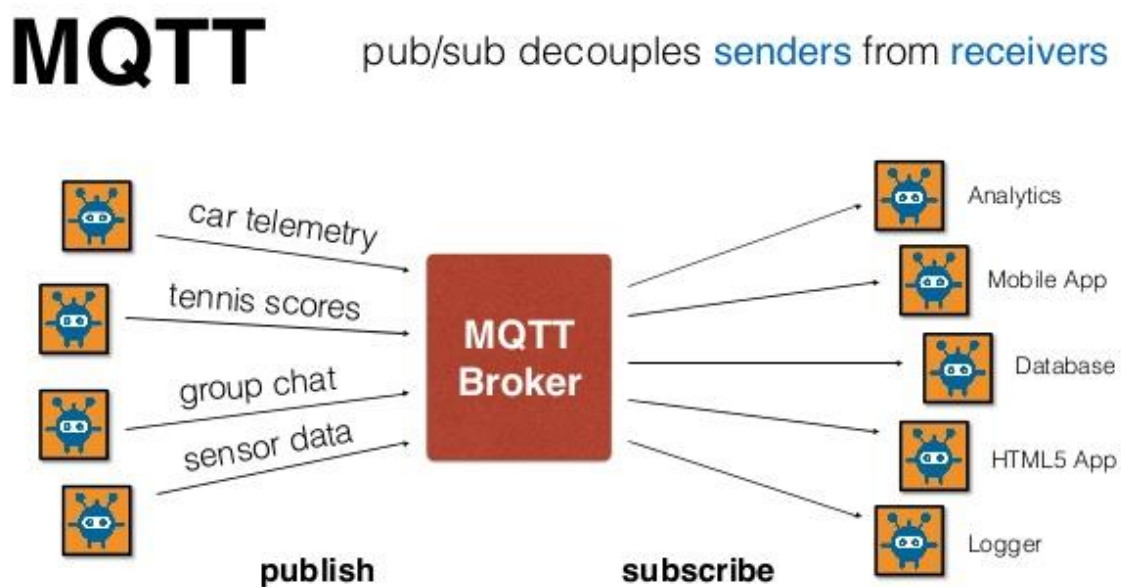
The publish and subscribe strategy is the first idea. A device can publish a message on a topic or subscribe to a certain topic in a publish and subscribe system to receive communications.

MQTT – Topics

The method you indicate where you want a message to be published or how you indicate your interest in incoming messages is through topics. Strings used to represent topics are divided into groups by a forward slash. A topic level is indicated by each forward slash.

MQTT – Broker

The MQTT broker is in charge of gathering all messages, filtering them, determining who is interested in them, and finally posting the message to all clients who have subscribed.



MQTT Quality of service types QoS 0 & 1 & 2

QoS 0 - at most once (Fire and forget)

Zero is the minimum QoS level. Best-effort delivery is ensured at this service level. No delivery assurance is provided. The communication is not kept or forwarded by the sender, and the recipient does not acknowledge receiving it.



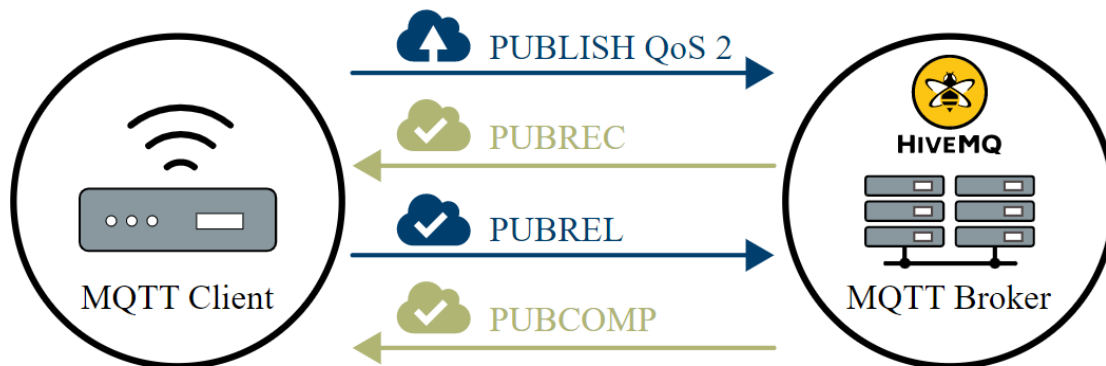
QoS 1 - at least once

A message will be delivered to the recipient at least once, according to QoS level 1. Until the recipient sends a PUBACK packet acknowledging receipt of the message, the sender holds onto the message. A message may be transmitted or delivered more than once.



QoS 2 - exactly once

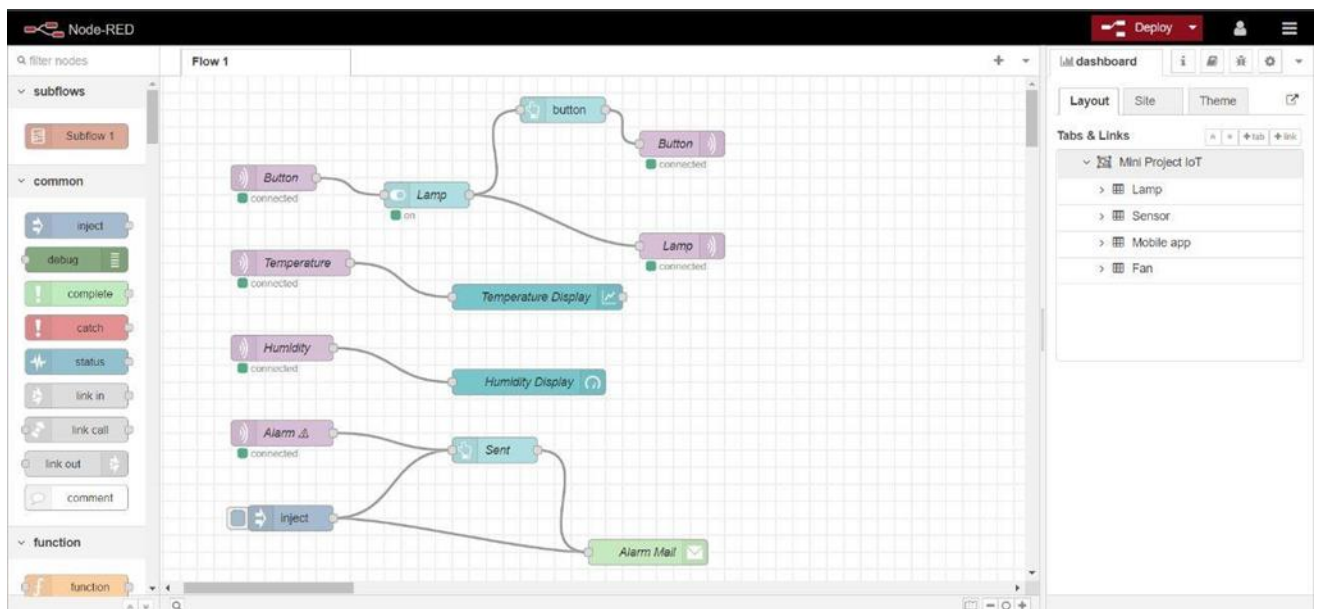
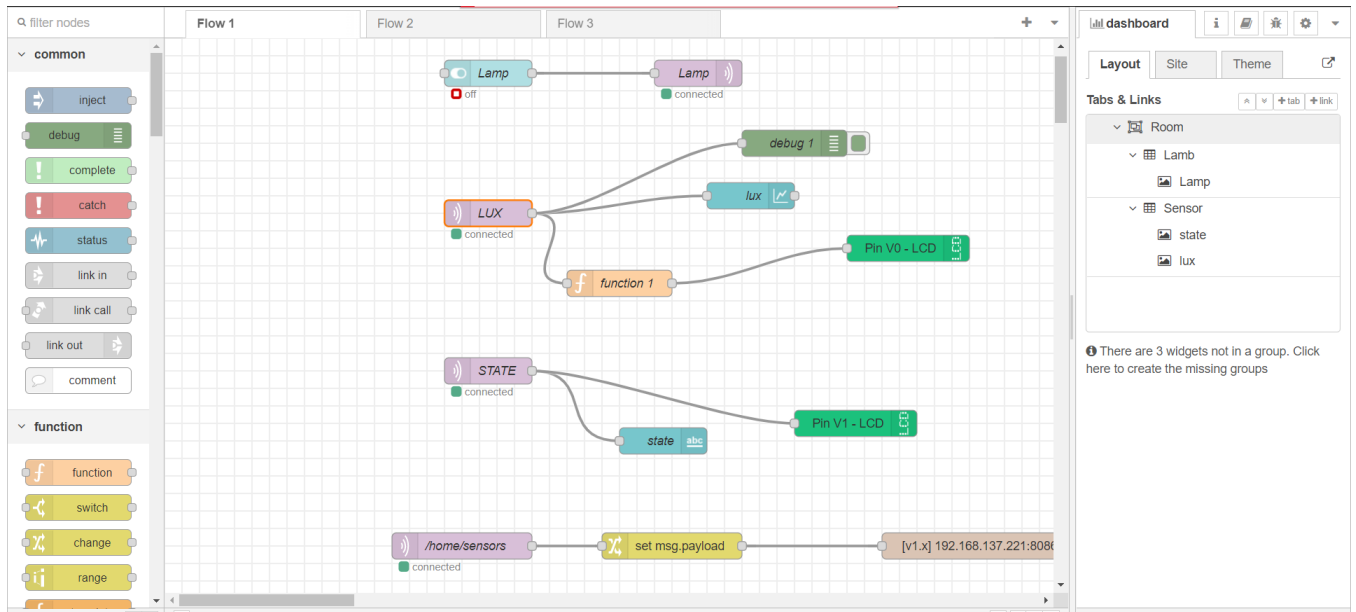
The highest level of service in MQTT is QoS 2. This level ensures that the intended recipients only receive each message once. The slowest and safest quality of service level is QoS 2. A minimum of two request/response flows (**a four-part handshake**) between the sender and the receiver are required to offer the guarantee. The initial PUBLISH message's packet identifier is used by the sender and receiver to coordinate message delivery.



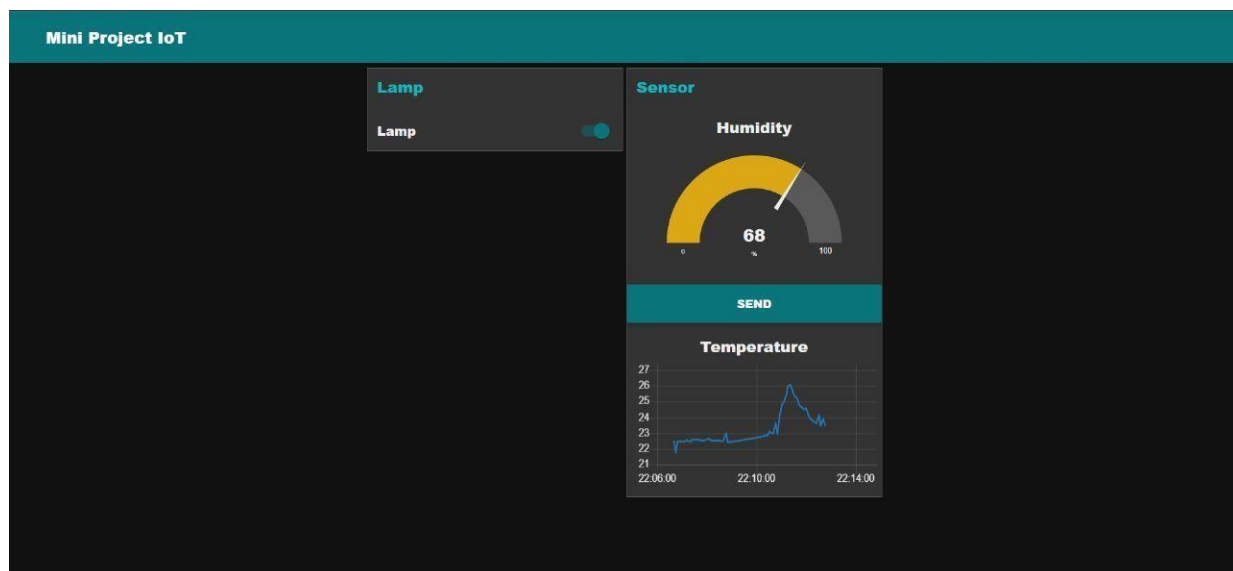
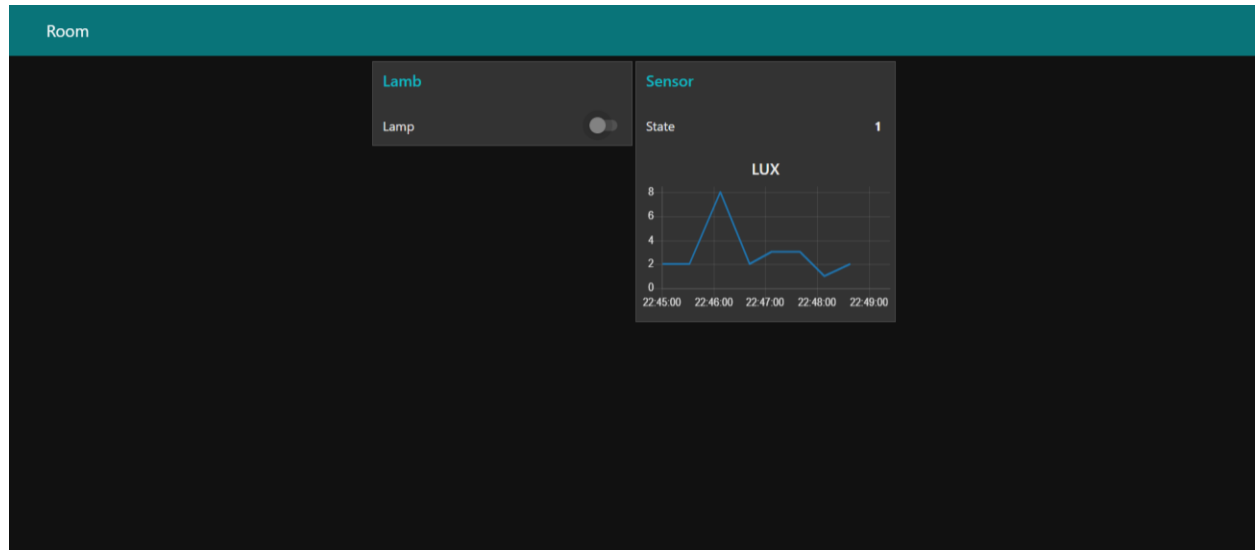
- Receivers receive QoS 2 PUBLISH packets from senders, process the publish message as necessary, and then send back an acknowledgement packet (**PUBREC**) **Publish received** to the sender.
- The initial PUBLISH packet can be securely discarded after the sender receives a PUBREC packet from the receiver. The sender replies with a (**PUBREL**) **Publish released** packet after storing the receiver's PUBREC packet.
- The receiver can discard any previously stored states after receiving the PUBREL packet and respond with a (**PUBCOMP**) **Publish complete** packet.

Node-red flow diagram:

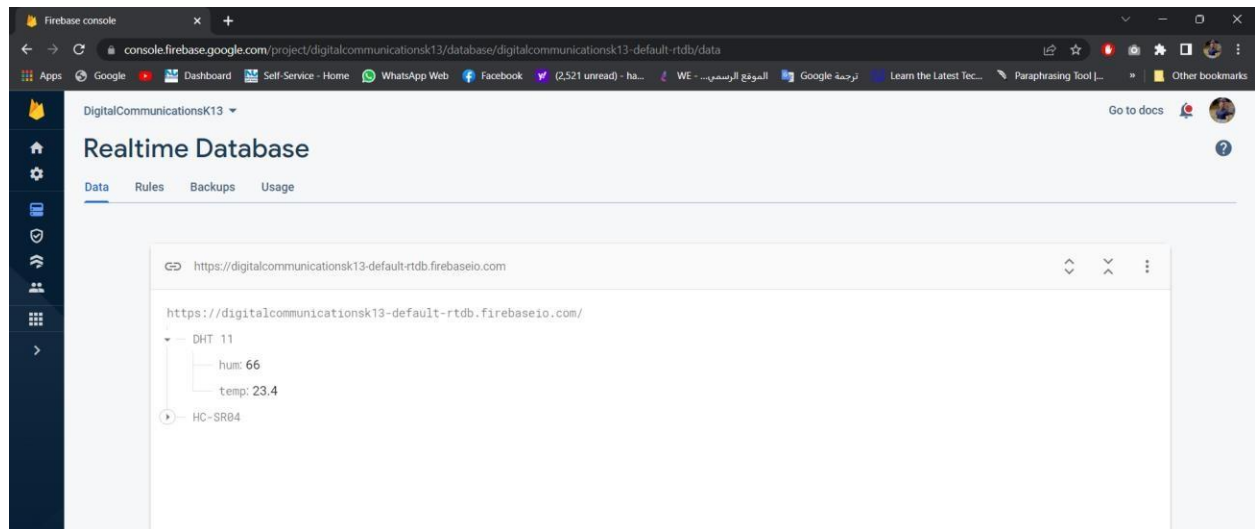
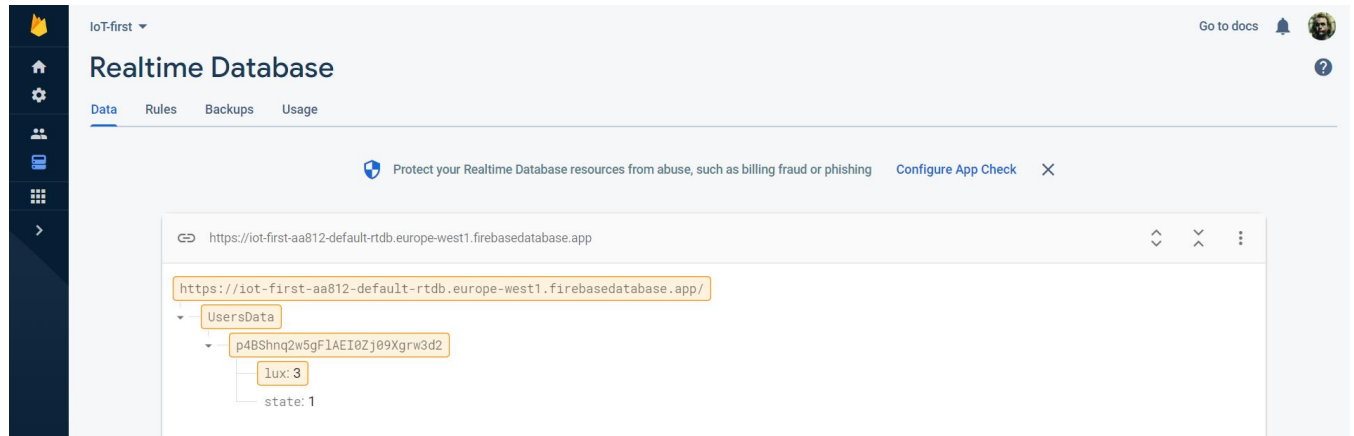
Node-red dashboard UI works with tabs and groups method where the room is the whole dashboard and there is a group for MQTT output nodes (lamp) and a group for MQTT input nodes (sensor) each dashboard is subscribed to a certain group





Node-red user interface:




Firebase Dashboard on the specified path (Topic)









Adafruit and Blynk

 [Devices](#) [Feeds](#) [Dashboards](#) [Actions](#) [Power-Ups](#)  [New Device](#)


ehelsayed / Feeds 


[New Feed](#) [New Group](#)



Feed Name	Key	Last value	Recorded
<input type="checkbox"/> gas	gas	1	less than a minute ago 
<input type="checkbox"/> humidity	humidity	37.0	less than a minute ago 
<input type="checkbox"/> moisture	moisture	0	2 minutes ago 
<input type="checkbox"/> temperature	temperature	24.0	less than a minute ago 



  My organization - 9295RS

[Back](#)

1 Device 

 • Smart Greenhouse

 **Smart Greenhouse** Offline 

 Adham  My organization - 9295RS [Add Tag](#)

[Dashboard](#) [Timeline](#) [Device Info](#) [Metadata](#) [Actions Log](#)

[Latest](#) [Last Hour](#) [6 Hours](#) [1 Day](#) [1 Week](#) [1 Month](#) [3 Months](#) [Custom](#)


Gas Sensor

Normal


Label Copy

Hydrated Soil


Mode

 Automatic


Temperature ...

 23 °C


Humidity Value

 41 ppm

Fan

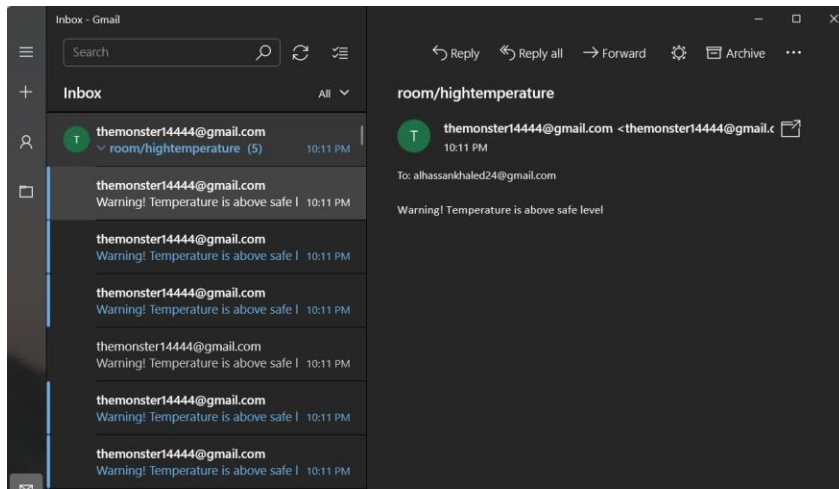
 ON

Water Pump

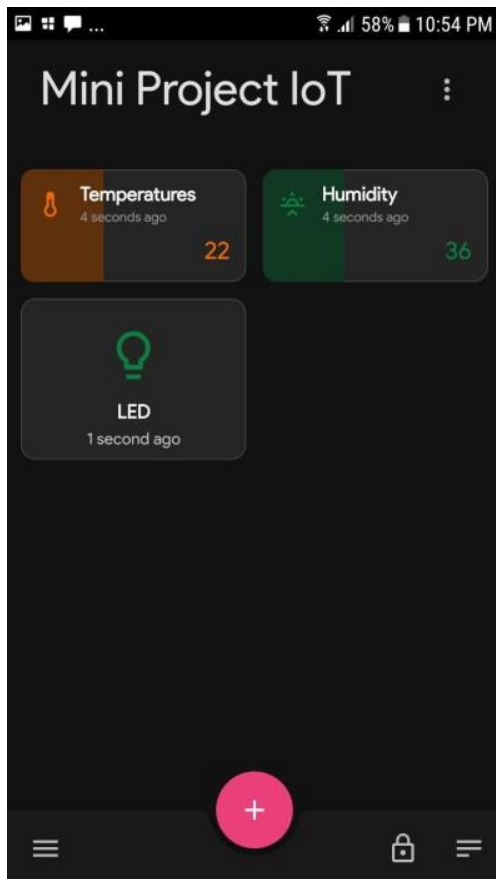
 ON

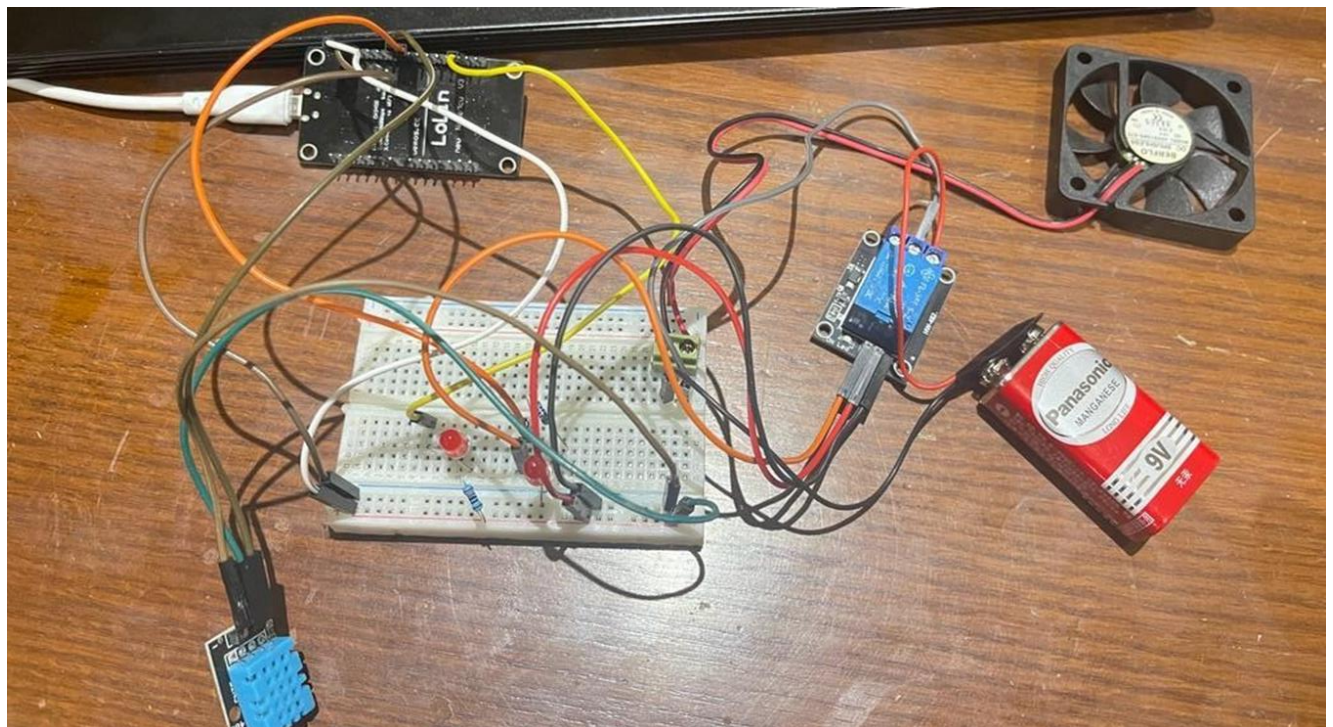
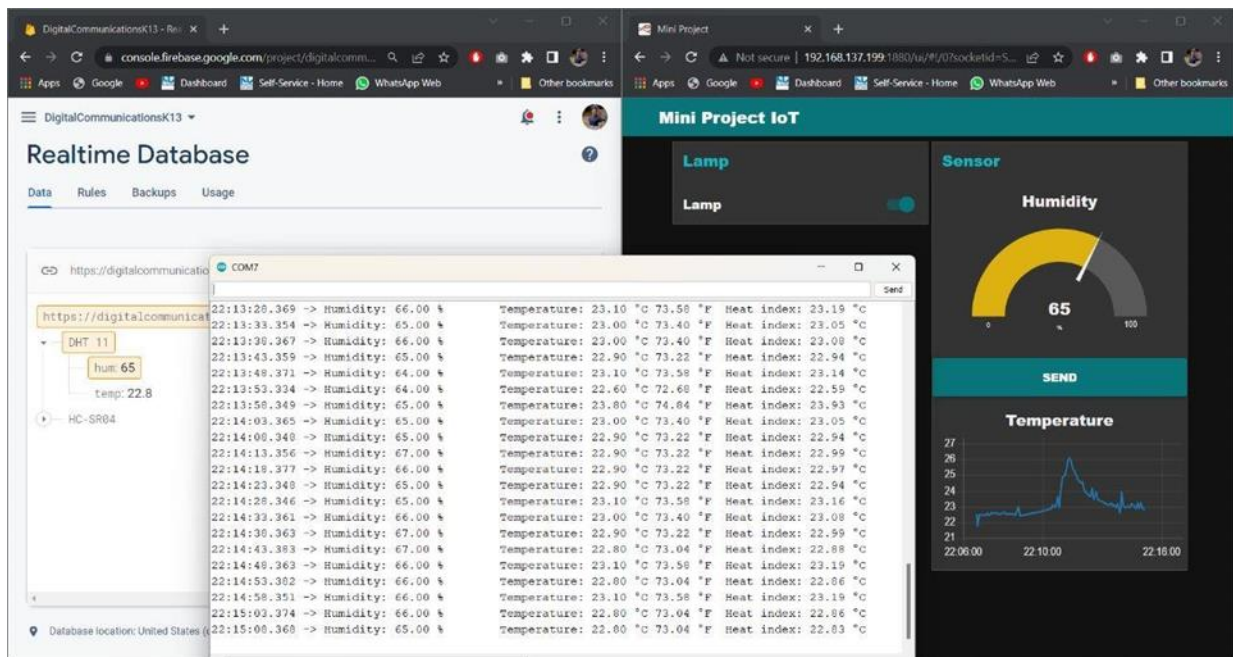
Region: ny3 [Privacy Policy](#)

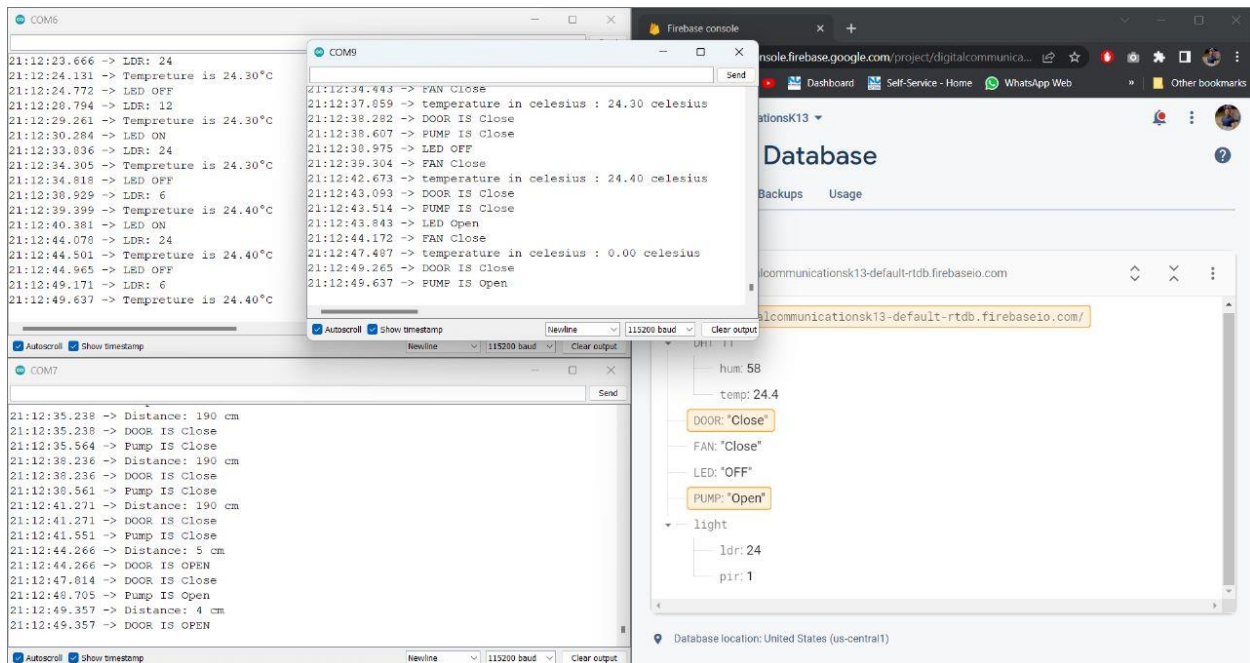
Email:



Mobile Application:

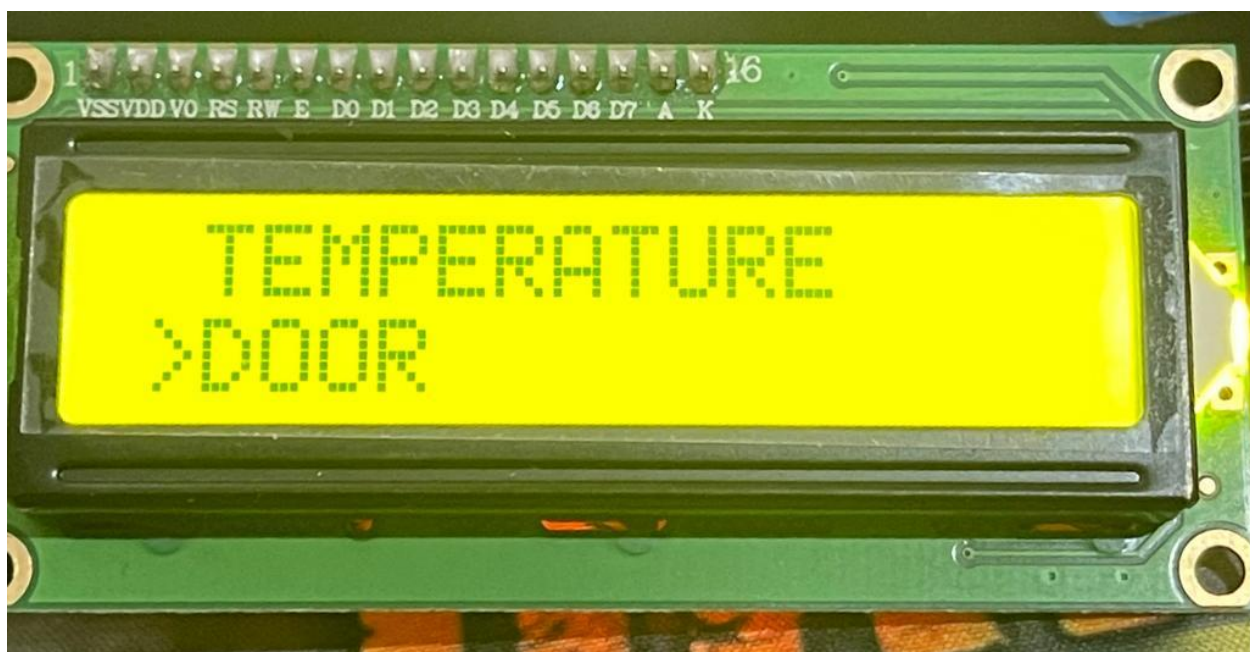


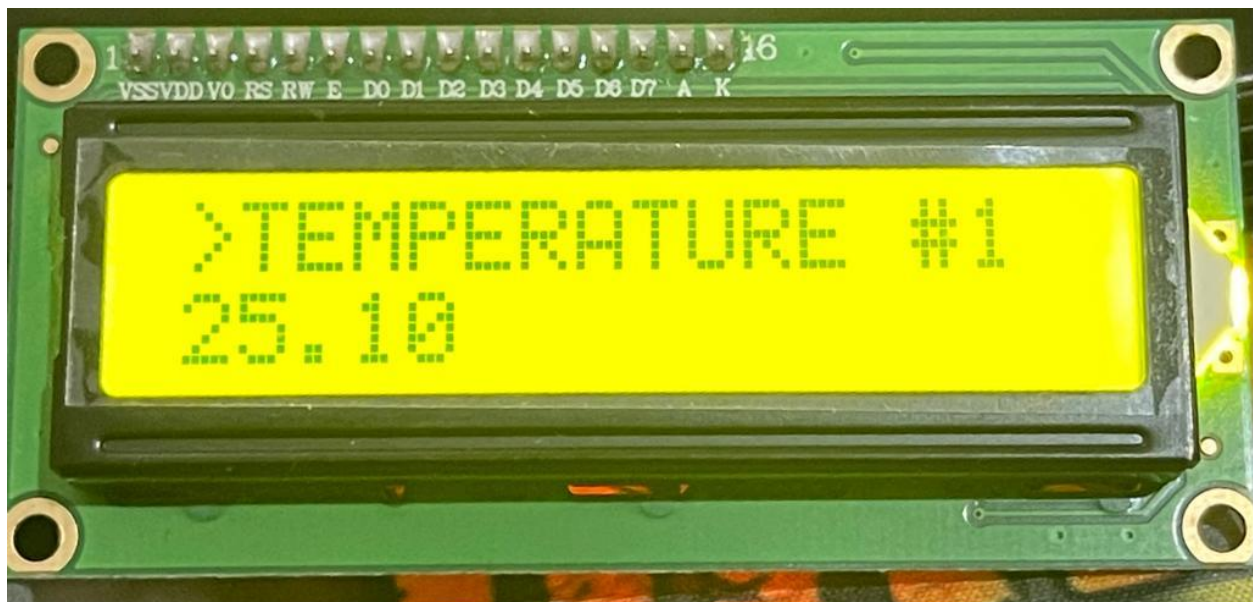




We made also a menu at lcd which can also show al the data wireless through

Wi-Fi





Appendix:

```
#if defined(ESP32) #include <WiFi.h> #elif defined(ESP8266)
```

```
#include <ESP8266WiFi.h> #include <PubSubClient.h> #endif
```

```
#include <Firebase_ESP_Client.h>
```

```
//dht
```

```
#include "DHT.h"
```

```
#include <Adafruit_Sensor.h> #include <espnow.h>
```

```
// Change the credentials below, so your ESP8266 connects to your router const char* ssid =  
"Hassan";
```

```
const char* password = "Aa123456790";
```

```
// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker const  
char* mqtt_server = "192.168.137.199";
```

```
//firebase token
```

```
//Provide the token generation process info. #include "addons/TokenHelper.h"
```

```

//Provide the RTDB payload printing info and other helper functions. #include
"addons/RTDBHelper.h"

// Insert Firebase project API Key

#define API_KEY "AIzaSyDOpG6ZPgepDOW5JqkgbEZYnuLkvR8RhY0"

// Insert RTDB URLdefine the RTDB URL */

#define DATABASE_URL "digitalcommunicationsk13-default-rtdb.firebaseio.com"

//Define Firebase Data object FirebaseData fbdo; //pointer

FirebaseAuth auth; FirebaseConfig config;

unsigned long sendDataPrevMillis = 0; int count = 0;

bool signupOK = false;

long duration; float temp; float hum;

#define DHTTYPE DHT11

// Uncomment one of the lines bellow for whatever DHT sensor type you're using! #define
DHTTYPE DHT11 // DHT 11

// Initializes the espClient. You should change the espClient name if you have multiple ESPs
running in your home automation system

WiFiClient espClient; PubSubClient client(espClient);

// DHT Sensor NodeMCU board #define DHTTYPE DHT11

```

```
// DHT Sensor
```

```
const int DHTPin = D6;
```

```
// Lamp - LED NodeMCU board const int lamp = D2;
```

```
// FAN - GPIO 0 = D3 on ESP-12E NodeMCU board const int fan = D5;
```

```
// Initialize DHT sensor.
```

```
DHT dht(DHTPin, DHTTYPE);
```

```
// Timers auxiliar variables long now = millis();
```

```
long lastMeasure = 0;
```

```
// Don't change the function below. This functions connects your ESP8266 to your router void
```

```
setup_wifi() {
```

```
delay(10);
```

```
// We start by connecting to a WiFi network Serial.println();

Serial.print("Connecting to "); Serial.println(ssid); WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) { delay(500);

Serial.print(".");

}

Serial.println("");

Serial.print("WiFi connected - ESP IP address: "); Serial.println(WiFi.localIP());

}
```

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to

// Change the function below to add logic to your program, so when a device publishes a message to a topic that

```
// your ESP8266 is subscribed you can actually do something void callback(String topic, byte*
message, unsigned int length) { Serial.print("Message arrived on topic: ");

Serial.print(topic); Serial.print(". Message: "); String messageTemp;
```



```
for (int i = 0; i < length; i++) { Serial.print((char)message[i]); messageTemp += (char)message[i];  
  
}
```

```
Serial.println();
```

```
// Feel free to add more if statements to control more GPIOs with MQTT
```

```
// If a message is received on the topic room/lamp, you check if the message is either on or off.
```

```
Turns the lamp GPIO according to the message if (topic == "room/lamp") { Serial.print("Changing  
Room lamp to ");
```

```
if (messageTemp == "true") { digitalWrite(lamp, HIGH); Serial.print("On");  
  
}
```

```
else if (messageTemp == "false") { digitalWrite(lamp, LOW); Serial.print("Off");  
  
}
```

```
}
```

```
Serial.println();
```

```
}
```

```

// This functions reconnects your ESP8266 to your MQTT broker

// Change the function below if you want to subscribe to more topics with your ESP8266 void
reconnect() {

// Loop until we're reconnected while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

// Attempt to connect

if (client.connect("ESP8266Client")) { Serial.println("connected");

// Subscribe or resubscribe to a topic

// You can subscribe to more topics (to control more LEDs in this example)
client.subscribe("room/lamp");

} else { Serial.print("failed, rc="); Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying delay(5000);

}

}

```

```
}
```

```
// The setup function sets your ESP GPIOs to Outputs, starts the serial communication at a baud  
rate of 115200
```

```
// Sets your mqtt broker and sets the callback function
```

```
// The callback function is what receives messages and actually controls the LEDs void setup() {
```

```
pinMode(lamp, OUTPUT); pinMode(fan, OUTPUT);
```

```
dht.begin();
```

```
Serial.begin(115200); setup_wifi();
```

```
client.setServer(mqtt_server, 1883); client.setCallback(callback);
```

```
/* Assign the api key (required) */ config.api_key = API_KEY;
```

```
/* Assign the RTDB URL (required) */ config.database_url = DATABASE_URL;
```

```

/* Sign up */

if (Firebase.signUp(&config, &auth, "", "")) { Serial.println("ok");

signupOK = true;

}

else {

Serial.printf("%s\n", config.signer.signupError.message.c_str()); // anonymous

}


/* Assign the callback function for the long running token generation task */
config.token_status_callback = tokenStatusCallback; //see addons/TokenHelper.h


Firebase.begin(&config, &auth); Firebase.reconnectWiFi(true);

}

// For this project, you don't need to change anything in the loop function. Basically it ensures that
you ESP is connected to your broker

```

```

void loop() {

if (Firebase.ready() && signupOK && (millis() - sendDataPrevMillis > 5000 ||
sendDataPrevMillis

== 0)) {

sendDataPrevMillis = millis();


if (!client.connected()) { reconnect();

}

if (!client.loop()) client.connect("ESP8266Client");


now = millis();

// Publishes new temperature and humidity every 30 seconds if (now - lastMeasure > 3000) {

lastMeasure = now;

// Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor) float h =
dht.readHumidity();

```

```

// Read temperature as Celsius (the default) float t = dht.readTemperature();

// Read temperature as Fahrenheit (isFahrenheit = true) float f = dht.readTemperature(true);


static char humidityTemp[7]; dtostrf(h, 6, 2, humidityTemp); if (t >= 24) {

digitalWrite(fan, HIGH);

Serial.print("FAN ON Sending an Email!! \n"); client.publish("room/hightemperature",
humidityTemp);

}

else {

digitalWrite(fan, LOW);


//Serial.print("FAN OFF");

}


// Check if any reads failed and exit early (to try again).

if (isnan(h) || isnan(t) || isnan(f)) { Serial.println("Failed to read from DHT sensor!"); return;

}

```

```
// Computes temperature values in Celsius float hic = dht.computeHeatIndex(t, h, false); static char  
temperatureTemp[7];
```

```
dtostrf(hic, 6, 2, temperatureTemp);
```

```
client.publish("room/temperature",    temperatureTemp);    client.publish("room/humidity",  
humidityTemp);
```

```
Serial.print("Humidity: "); Serial.print(h);
```

```
Serial.print(" %\t Temperature: "); Serial.print(t);
```

```
Serial.print(" °C "); Serial.print(f);
```

```
Serial.print(" °F\t Heat index: "); Serial.print(hic);
```

```
Serial.println(" °C ");
```

```
}
```

```
// Write an Int number on the database path test/int
```

```
if (Firebase.RTDB.setFloat(&fbdo, "DHT 11/hum", dht.readHumidity())) {
```

```
}
```

```
else { Serial.println("FAILED");

Serial.println("REASON: " + fbdo.errorReason());

}

// Write an Float number on the database path test/float

if (Firebase.RTDB.setFloat(&fbdo, "DHT 11/temp", dht.readTemperature() )) {

}

else { Serial.println("FAILED");

Serial.println("REASON: " + fbdo.errorReason());

}

delayMicroseconds(300);

}

}
```

Project video [LINK](#)