ECE363 Project Report

On

Weather station using NodeMCU and working of CoAP

B-TECH MECHATRONICS

LOVELY PROFESSIONAL

UNIVERSITY PHAGWARA, PUNJAB

From 07/03/23 to 21/04/23

SUBMITTED

BY

Amith S

SUBMITTED

TO

Dr. Kesavan Gopal

STUDENT DECLARATION

To whom so ever it may concern

I, **Amith S,** hereby declare that the work done by me on "**Weather station using NodeMCU and working of CoAP**" from March 2023 to April 2023, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **B-Tech in Mechatronics**.

Amith S (11901789)

[illegible]

Date:21-04-2023

ACKNOWLEDGEMENT

Working on the Project Weather station using NodeMCU and working of CoAP was interesting. I have kept my full efforts in this project to make it successful. I have learnt a lot during working on this project.

I pay my deep sense of gratitude to **Dr. Kesavan Gopal**, to encourage me to the highest peak and to provide me the opportunity to work on this Project. I am immensely obliged to my friends for their elevating for their elevating inspiration, encouraging guidance and kind supervision in the completion of my project.

After completing this Project, I feel more ready to explore IOT Projects using Node MCU and working of CoAP protocol. This course helped me to grasp the field of iot and helped me understand how to operate and work on sensors and different types of communication protocol.

LOVELY PROFESSIONAL UNIVERSITY

Index

INTRODUCTION

In today's world, where global warming and climate change have become critical issues, monitoring the weather and air quality has become a necessity. A weather station is a device that helps to monitor various weather parameters such as temperature, humidity, pressure, wind speed, and direction. It can be used to predict and analyze weather patterns, which is essential for agriculture, aviation, and other industries.

In this modern era of the Internet of Things (IoT), creating a weather station using sensors and microcontrollers has become relatively easy and affordable. In this project, we will use the DHT11 temperature and humidity sensor, MQ135 air quality sensor, and NodeMCU board, which is an IoT development board, to create a weather station.

The DHT11 sensor is a low-cost digital sensor that can measure temperature and humidity with reasonable accuracy. It has a simple two-wire interface and is widely used in weather stations and HVAC systems. The MQ135 sensor is a gas sensor that can detect the presence of harmful gases such as carbon dioxide, ammonia, and benzene.

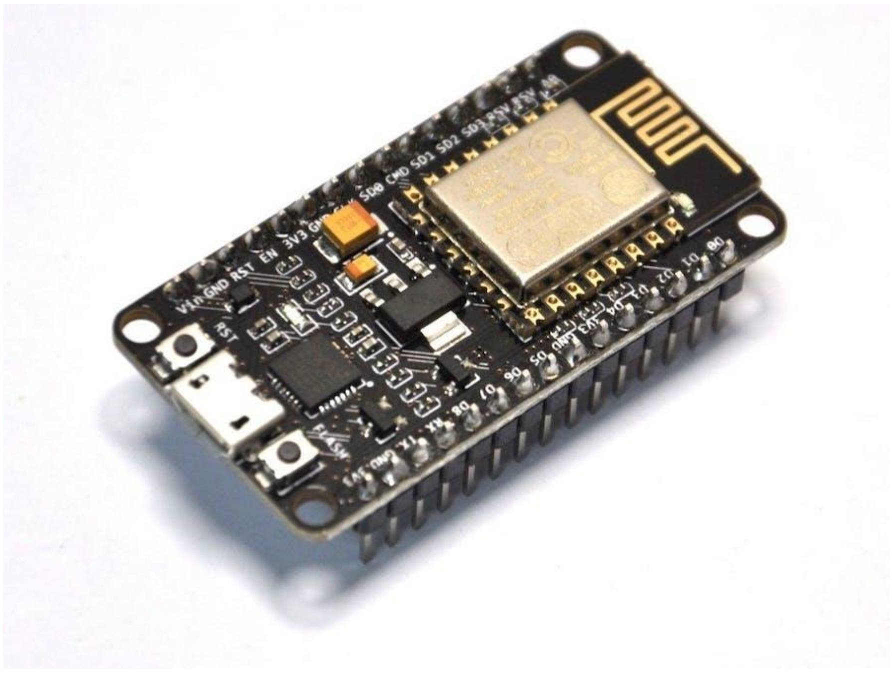
The NodeMCU board is a popular Wi-Fi-enabled development board based on the ESP8266 chip. It has built-in support for the Lua programming language and can be programmed using the Arduino IDE. It has several input/output pins that can be used to interface with sensors and other peripherals.

To make our weather station accessible over the internet, we will use the Constrained Application Protocol (CoAP). CoAP is a lightweight protocol designed for IoT devices that have limited computing resources and network bandwidth. It is similar to HTTP but uses UDP instead of TCP, which reduces overhead and improves scalability.

Overall, this project will demonstrate how to create a simple and affordable weather station using commonly available sensors and microcontrollers. By using CoAP, we will be able to make the weather station accessible over the internet, allowing us to monitor weather and air quality data from anywhere in the world.

COMPONENTS

NodeMCU

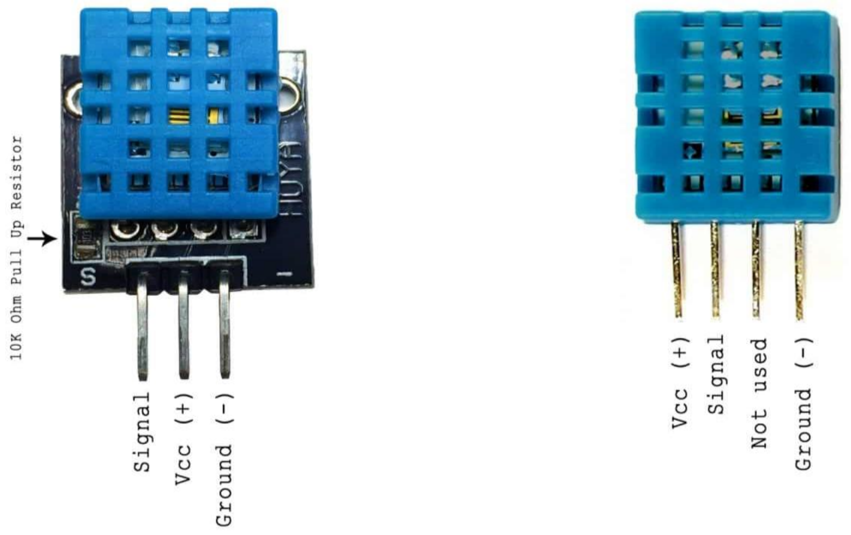

Vin GND RST EN 3V3 GND
SD8 CMD SD1 SD2 SD3 RSV RSV A0
RST
FLASH
D0 D1 D2 D3 D4 3V3 GND D5 D6 D7 D8 RX TX GND 3V3

NodeMCU is an open-source firmware and development board based on the ESP8266 Wi-Fi module. The ESP8266 is a low-cost Wi-Fi-enabled microcontroller chip that has become popular in the IoT community due to its affordability and ease of use. NodeMCU is designed to make it easier for developers to prototype and build IoT applications using the ESP8266 chip.

Specification

- Microcontroller: ESP8266-12E module with Tensilica Xtensa LX106 processor
- Clock speed: 80MHz (can be overclocked up to 160MHz)
- Flash memory: 4MB (32Mb) integrated SPI flash
- Wi-Fi: 802.11 b/g/n Wi-Fi connectivity with WPA/WPA2 encryption
- GPIO: 17 GPIO pins, including 1 analog input pin (3.2V max input)
- ADC: 10-bit ADC with a voltage range of 0 to 3.3V
- Programming: Lua scripting language support, NodeMCU API for GPIO, PWM, I2C, and 1-Wire
- Operating voltage: 3.3V DC (powered via USB or external power source)
- USB: Micro USB for power and serial communication (via CP2102 USB-to-UART bridge)
- Dimensions: 49mm x 24mm x 13mm

DHT 11



The DHT11 sensor is a simple and inexpensive sensor that provides basic temperature and humidity sensing capabilities. While it may not be as accurate or precise as some other sensors, it is still a useful tool for many applications. The single-wire digital interface and low power consumption make it easy to use with microcontrollers and other electronic devices.

Specification

- Operating voltage: 3 to 5.5 volts DC
- Temperature range: 0°C to 50°C with an accuracy of ±2°C
- Humidity range: 20% to 80% with an accuracy of ±5%
- Output: Single-wire digital interface with a resolution of 1°C and 1% RH
- Refresh rate: 1Hz
- Response time: 2 seconds
- Dimensions: 12mm x 15.5mm x 5.5mm
- Weight: 1.8 grams

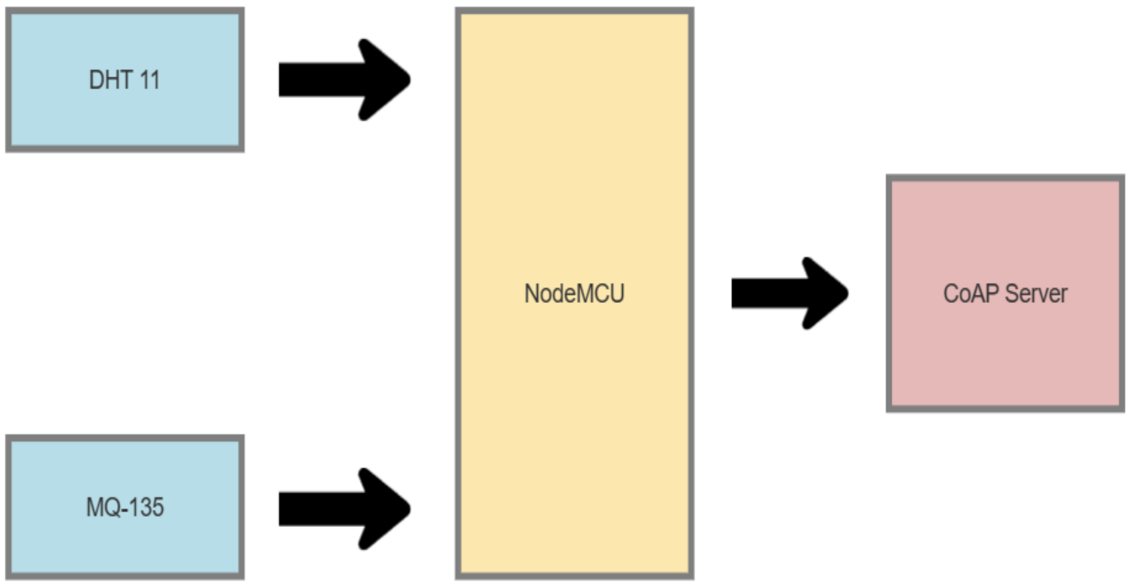
MQ-135

The MQ-135 gas sensor is a popular choice for air quality monitoring applications due to its low cost, versatility, and ease of use. Its detection range covers a wide range of common air pollutants, making it a useful tool for monitoring indoor and outdoor air quality. The adjustable sensitivity allows users to customize the sensor's response to their specific needs. The analog output makes it easy to use with microcontrollers and other electronic devices.

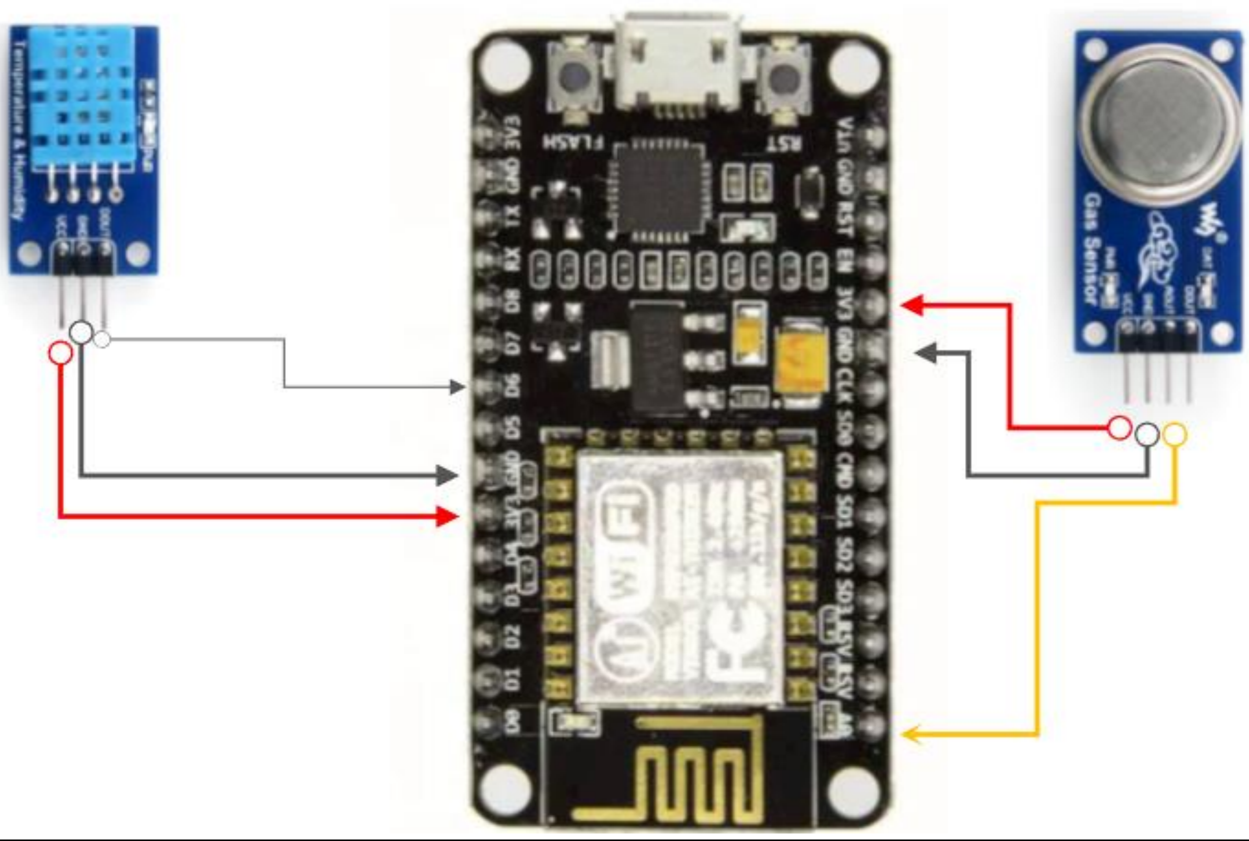
Specification

- Operating voltage: 5V DC
- Detection range:
    - Ammonia (NH_3): 10 to 300 ppm
    - Nitrogen oxides (NO_x): 10 to 1000 ppm
    - Benzene: 1 to 50 ppm
    - Toluene: 1 to 50 ppm
    - Alcohol: 10 to 500 ppm
    - Carbon Monoxide (CO): 10 to 500 ppm
    - Smoke: 1 to 10 mg/m3
- Sensitivity can be adjusted using a potentiometer.
- Analog output
- Dimensions: 36mm x 22mm x 25mm
- Weight: 5 grams

BLOCK DIAGRAM



CIRCUIT DIAGRAM

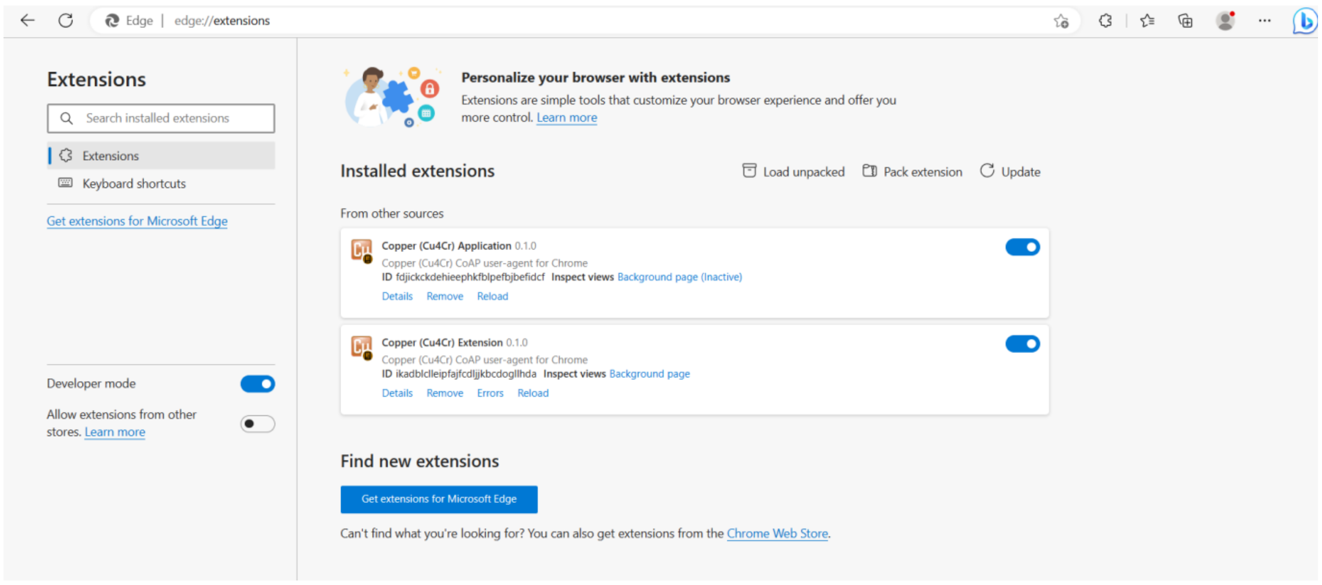


COAP SERVER SETUP

Copper (Cu4Cr) is a Chrome extension that allows users to send and receive CoAP messages using their web browser. CoAP stands for Constrained Application Protocol, which is a specialized web protocol designed for resource-constrained devices and networks.

- Install the Copper (Cu4Cr) extension from the Chrome Web Store.
- Once installed, click on the Copper (Cu4Cr) icon in your Chrome browser toolbar to open the extension.
- In the Copper (Cu4Cr) interface, enter the IP address or hostname of the CoAP server you wish to connect to.
- Select the desired CoAP method (e.g. GET, PUT, POST, DELETE) from the drop-down menu.
- Enter the CoAP URI path of the resource you wish to access.
- Click on the "Send" button to send the CoAP message to the server.
- The server's response will be displayed in the Copper (Cu4Cr) interface.

Images



Edge | edge://extensions

Extensions

Search installed extensions

Extensions

Keyboard shortcuts

Get extensions for Microsoft Edge

Developer mode

Allow extensions from other stores. Learn more

Personalize your browser with extensions

Extensions are simple tools that customize your browser experience and offer you more control. Learn more

Installed extensions

Load unpacked Pack extension Update

From other sources

Copper (Cu4Cr) Application 0.1.0
Copper (Cu4Cr) CoAP user-agent for Chrome
ID fdjickckdehieephkfblpefbjbefidcf **Inspect views** Background page (Inactive)
Details Remove Reload

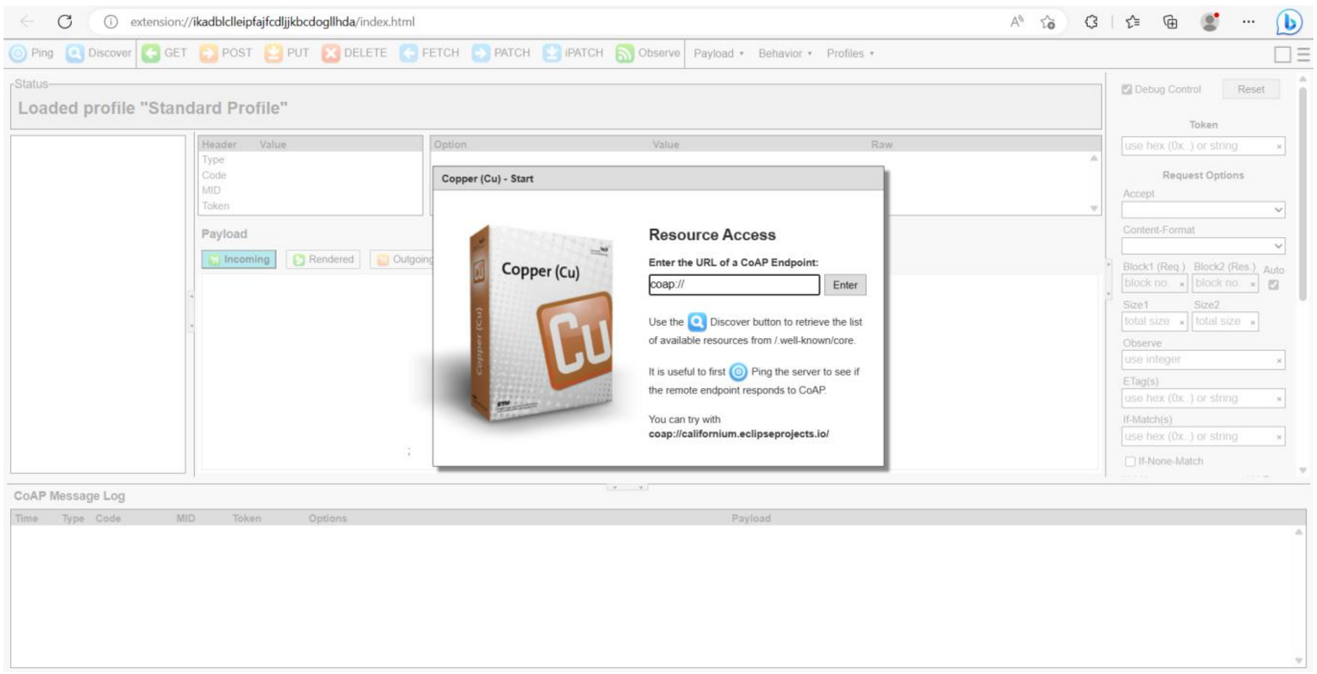
Copper (Cu4Cr) Extension 0.1.0
Copper (Cu4Cr) CoAP user-agent for Chrome
ID ikadblclleipfajfcdljjkbcdogllhda **Inspect views** Background page
Details Remove Errors Reload

Find new extensions

Get extensions for Microsoft Edge

Can't find what you're looking for? You can also get extensions from the Chrome Web Store.

Type the IP address.


Status
Loaded profile "Standard Profile"

Copper (Cu) - Start

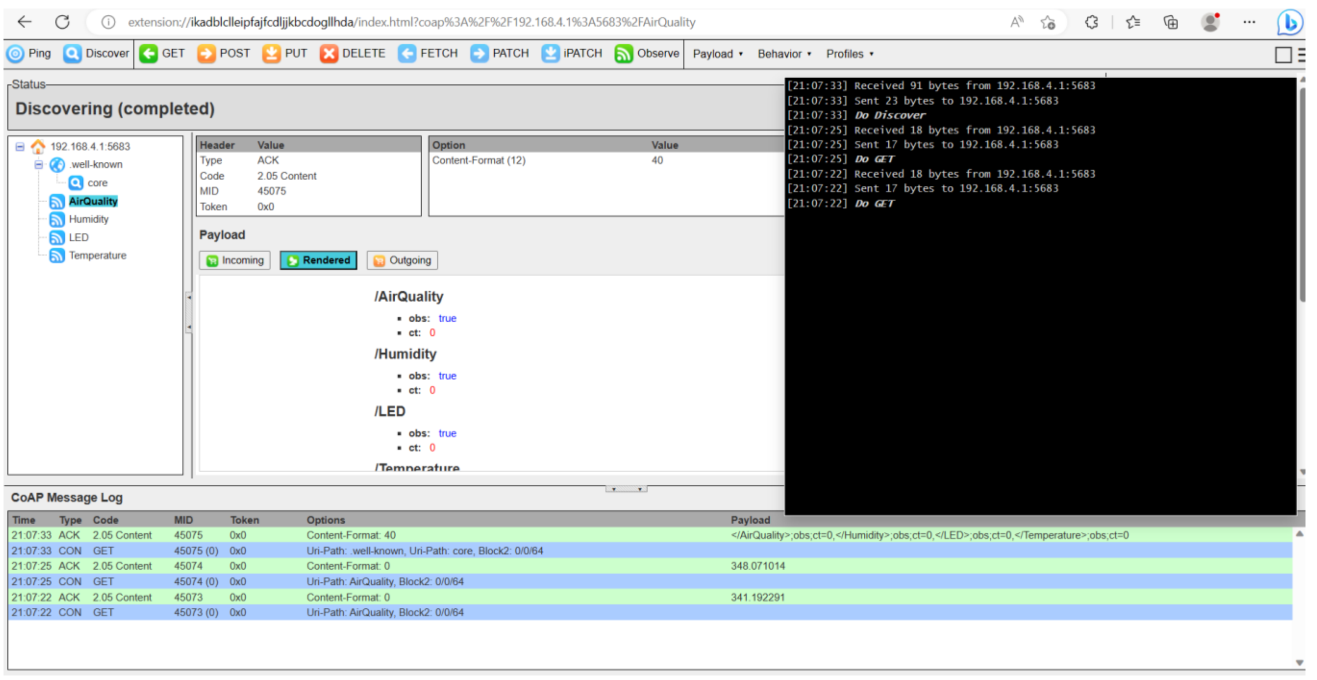
Resource Access

Enter the URL of a CoAP Endpoint:
coap://

Use the Discover button to retrieve the list of available resources from /.well-known/core.

It is useful to first Ping the server to see if the remote endpoint responds to CoAP.

You can try with
coap://californium.eclipseprojects.io/


extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FAirQuality

Status
Discovering (completed)

Header	Value
Type	ACK
Code	2.05 Content
MID	45075
Token	0x0

Option	Value
Content-Format (12)	40

Payload

/AirQuality
- obs: true
- ct: 0

/Humidity
- obs: true
- ct: 0

/LED
- obs: true
- ct: 0

/Temperature

```
[21:07:33] Received 91 bytes from 192.168.4.1:5683
[21:07:33] Sent 23 bytes to 192.168.4.1:5683
[21:07:33] Do Discover
[21:07:25] Received 18 bytes from 192.168.4.1:5683
[21:07:25] Sent 17 bytes to 192.168.4.1:5683
[21:07:25] Do GET
[21:07:22] Received 18 bytes from 192.168.4.1:5683
[21:07:22] Sent 17 bytes to 192.168.4.1:5683
[21:07:22] Do GET
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
21:07:33	ACK	2.05 Content	45075	0x0	Content-Format: 40	</AirQuality>;obs;ct=0,</Humidity>;obs;ct=0,</LED>;obs;ct=0,</Temperature>;obs;ct=0
21:07:33	CON	GET	45075 (0)	0x0	Uri-Path: .well-known, Uri-Path: core, Block2: 0/0/64	
21:07:25	ACK	2.05 Content	45074	0x0	Content-Format: 0	348.071014
21:07:25	CON	GET	45074 (0)	0x0	Uri-Path: AirQuality, Block2: 0/0/64	
21:07:22	ACK	2.05 Content	45073	0x0	Content-Format: 0	341.192291
21:07:22	CON	GET	45073 (0)	0x0	Uri-Path: AirQuality, Block2: 0/0/64	

CODE

Header File Used

```
#include <ESP8266WiFi.h>  // for NodeMCU to connect to Wifi

#include <DHT.h> // library to read temperature and humidity from Dht11 semsor

#include <MQ135.h> // library to convert analog to digital form
```

Declare our CoAP server and the packet handler

```
Thing::CoAP::Server server;

Thing::CoAP::ESP::UDPPacketProvider udpProvider;
```

Wifi detail to Esp8266 to connect to network

```
char* ssid = "*****";

char* password = "******";
```

Defining pin of dht11 and mq135 sensor

```
#define DHTPIN D6

#define PIN_MQ135 A0

#define LED 2
```

Initialize DHT sensor

```
DHT dht(DHTPIN, DHT11);

MQ135 mq135_sensor(PIN_MQ135);
```

Create an endpoint

```
server.CreateResource("Temperature", Thing::CoAP::ContentFormat::TextPlain, true) // True means that this resource is observable

   .OnGet([](Thing::CoAP::Request & request) { // We are here configuring telling our server that, when we receive a "GET" request to this endpoint, run the following code

     Serial.println("GET Request received for endpoint 'Temperature'");
```

Read the temperature from the DHT11 sensor

```
float temperature = dht.readTemperature();
    if (isnan(temperature)) {
      Serial.println("Failed to read temperature from DHT sensor!");
      return Thing::CoAP::Status::InternalServerError();
```

Return the current temperature

```
std::string result = std::to_string(temperature);
    return Thing::CoAP::Status::Content(result);
```

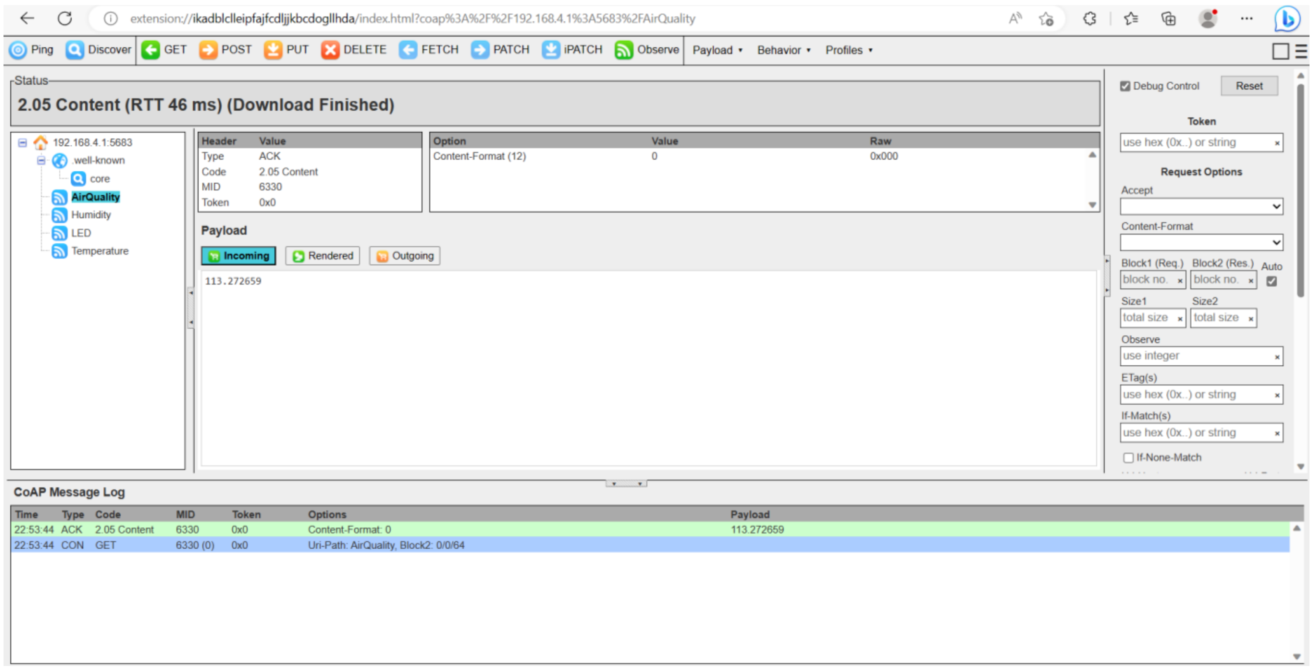
This code sets up an observable CoAP resource "LED" that responds to "GET" requests by reading the state of an LED pin and returning its current status as "On" or "Off".

```
server.CreateResource("LED", Thing::CoAP::ContentFormat::TextPlain, true) //True means
that this resource is observable
   .OnGet([](Thing::CoAP::Request & request) { //We are here configuring telling our server
that, when we receive a "GET" request to this endpoint, run the the following code
    Serial.println("GET Request received for endpoint 'LED'");

    //Read the state of our led.
    std::string result;
    if(digitalRead(LED) == HIGH)
      result = "Off";
    else
      result = "On";

     //Return the current state of our "LED".
    return Thing::CoAP::Status::Content(result);
```

This code sets up a CoAP resource that responds to "POST" requests by reading the payload message and toggling an LED pin on or off depending on the message, and returning "Created" with a message "Command Executed" if successful or "BadRequest" if the message is invalid.

```
.OnPost([](Thing::CoAP::Request& request) {  //We are here configuring telling our server that, when we receive a "POST" request to this endpoint, run the the following code
    Serial.println("POST Request received for endpoint 'LED'");

    //Get the message sent fromthe client and parse it to a string
    auto payload = request.GetPayload();
    std::string message(payload.begin(), payload.end());

    Serial.print("The client sent the message: ");
    Serial.println(message.c_str());

    if(message == "On") { //If the message is "On" we will turn the LED on.
      digitalWrite(LED, LOW);
    } else if (message == "Off") { //If it is "Off" we will turn the LED off.
      digitalWrite(LED, HIGH);
    } else { //In case any other message is received we will respond a "BadRequest" error.
      return Thing::CoAP::Status::BadRequest();
    }

    //In case "On" or "Off" was received, we will return "Ok" with a message saying "Command Executed".
    return Thing::CoAP::Status::Created("Command Executed");
```

COAP SERVER DASHBOARD



extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FAirQuality

Status

2.05 Content (RTT 46 ms) (Download Finished)

Header	Value
Type	ACK
Code	2.05 Content
MID	6330
Token	0x0

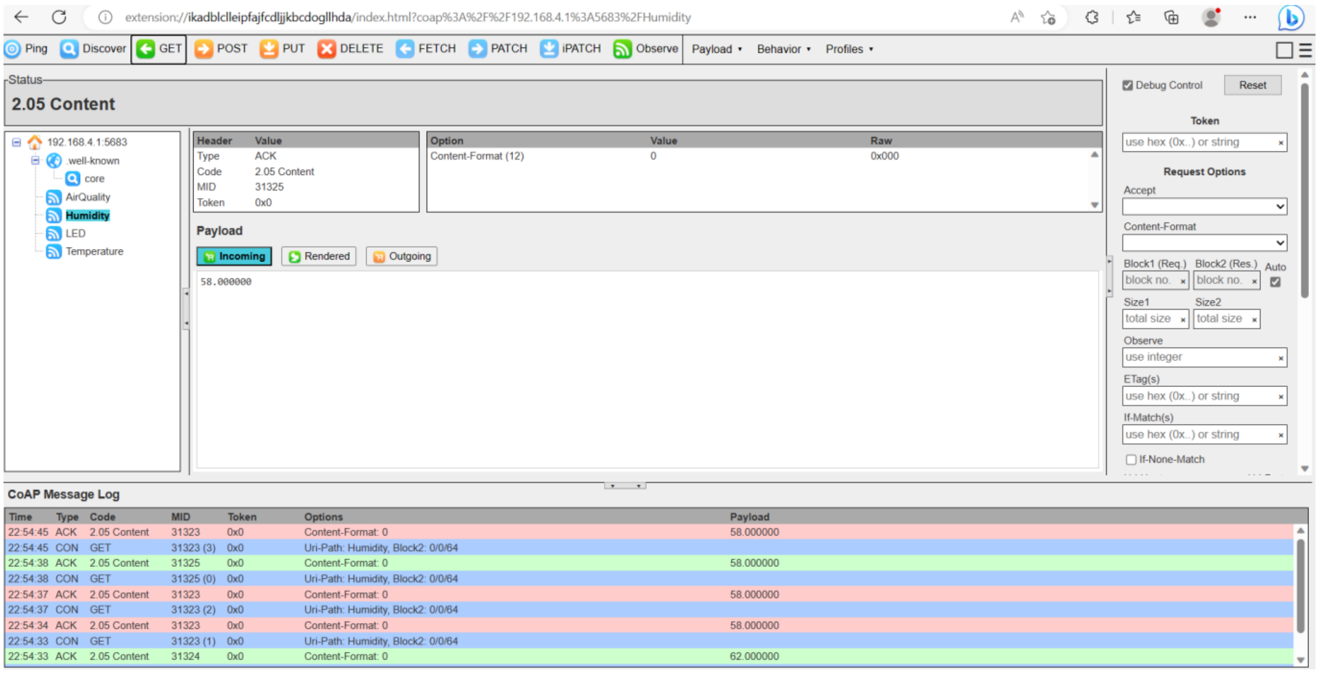
Option	Value	Raw
Content-Format (12)	0	0x000

Payload

```
113.272659
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
22:53:44	ACK	2.05 Content	6330	0x0	Content-Format: 0	113.272659
22:53:44	CON	GET	6330 (0)	0x0	Uri-Path: AirQuality, Block2: 0/0/64	



extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FHumidity

Status

2.05 Content

Header	Value
Type	ACK
Code	2.05 Content
MID	31325
Token	0x0

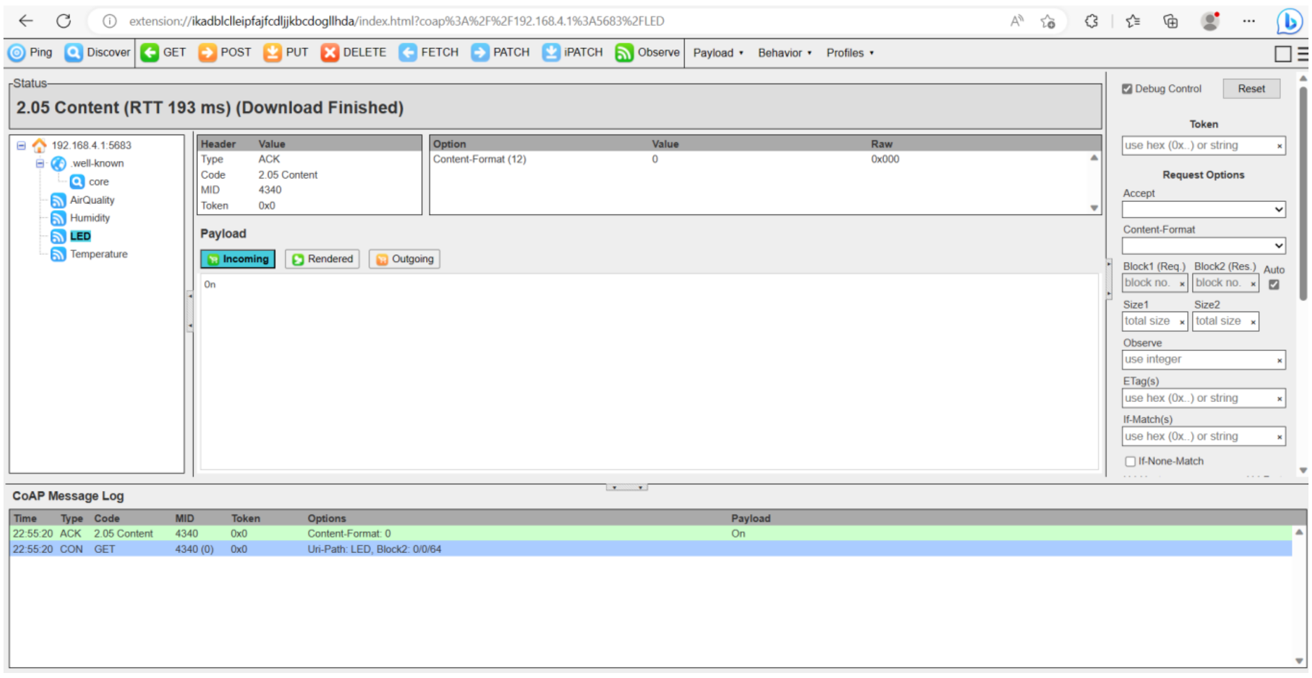
Option	Value	Raw
Content-Format (12)	0	0x000

Payload

```
58.000000
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
22:54:45	ACK	2.05 Content	31323	0x0	Content-Format: 0	58.000000
22:54:45	CON	GET	31323 (3)	0x0	Uri-Path: Humidity, Block2: 0/0/64	
22:54:38	ACK	2.05 Content	31325	0x0	Content-Format: 0	58.000000
22:54:38	CON	GET	31325 (0)	0x0	Uri-Path: Humidity, Block2: 0/0/64	
22:54:37	ACK	2.05 Content	31323	0x0	Content-Format: 0	58.000000
22:54:37	CON	GET	31323 (2)	0x0	Uri-Path: Humidity, Block2: 0/0/64	
22:54:34	ACK	2.05 Content	31323	0x0	Content-Format: 0	58.000000
22:54:33	CON	GET	31323 (1)	0x0	Uri-Path: Humidity, Block2: 0/0/64	
22:54:33	ACK	2.05 Content	31324	0x0	Content-Format: 0	62.000000



extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FLED

Status

2.05 Content (RTT 193 ms) (Download Finished)

Header	Value
Type	ACK
Code	2.05 Content
MID	4340
Token	0x0

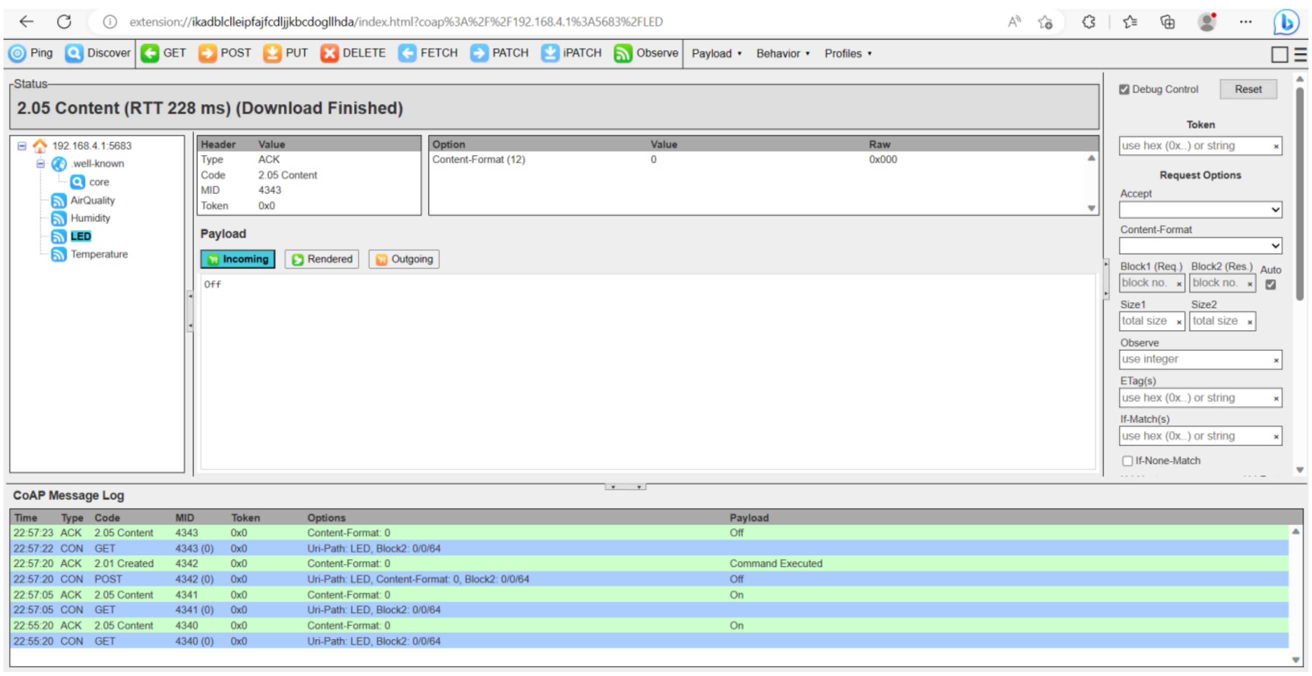
Option	Value	Raw
Content-Format (12)	0	0x000

Payload

```
On
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
22:55:20	ACK	2.05 Content	4340	0x0	Content-Format: 0	On
22:55:20	CON	GET	4340 (0)	0x0	Uri-Path: LED, Block2: 0/0/64	



extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FLED

Status

2.05 Content (RTT 228 ms) (Download Finished)

Header	Value
Type	ACK
Code	2.05 Content
MID	4343
Token	0x0

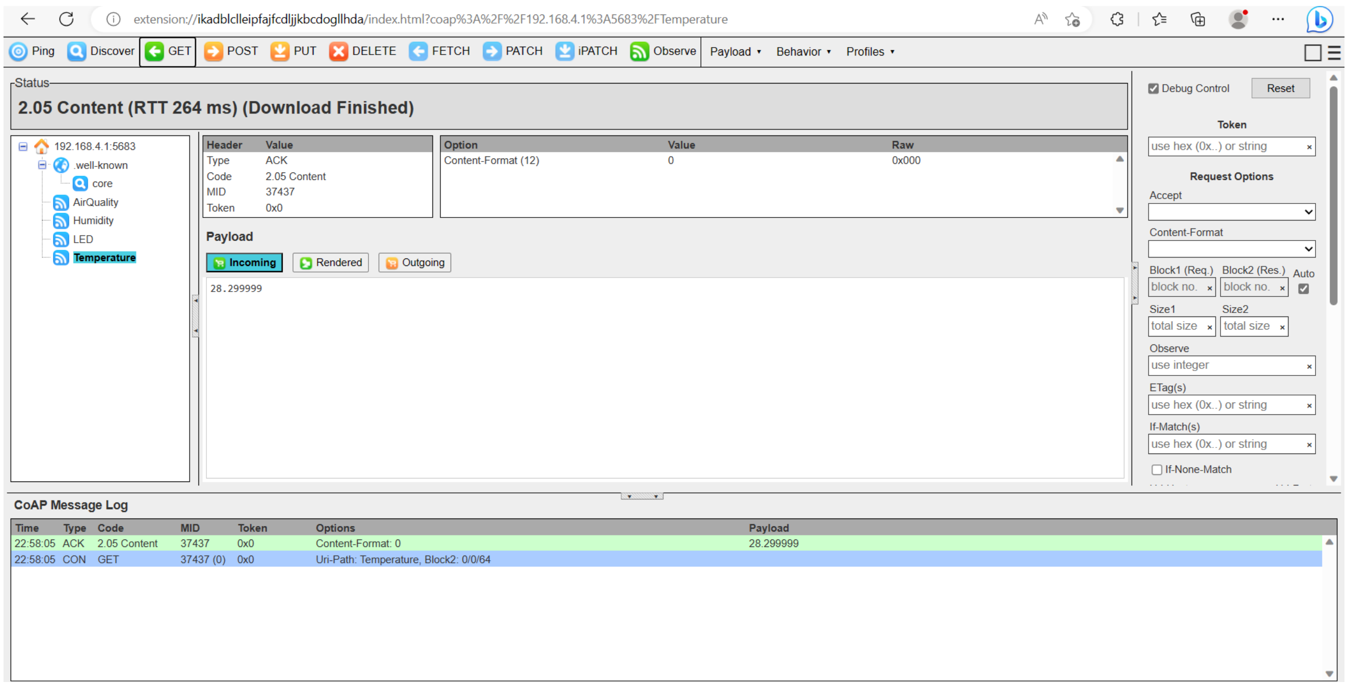
Option	Value	Raw
Content-Format (12)	0	0x000

Payload

```
Off
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
22:57:23	ACK	2.05 Content	4343	0x0	Content-Format: 0	Off
22:57:22	CON	GET	4343 (0)	0x0	Uri-Path: LED, Block2: 0/0/64	
22:57:20	ACK	2.01 Created	4342	0x0	Content-Format: 0	Command Executed
22:57:20	CON	POST	4342 (0)	0x0	Uri-Path: LED, Content-Format: 0, Block2: 0/0/64	Off
22:57:05	ACK	2.05 Content	4341	0x0	Content-Format: 0	On
22:57:05	CON	GET	4341 (0)	0x0	Uri-Path: LED, Block2: 0/0/64	
22:55:20	ACK	2.05 Content	4340	0x0	Content-Format: 0	On
22:55:20	CON	GET	4340 (0)	0x0	Uri-Path: LED, Block2: 0/0/64	



extension://ikadblclleipfajfcdljjkbcdogllhda/index.html?coap%3A%2F%2F192.168.4.1%3A5683%2FTemperature

Status

2.05 Content (RTT 264 ms) (Download Finished)

- 192.168.4.1:5683
  - .well-known
    - core
  - AirQuality
  - Humidity
  - LED
  - Temperature

Header	Value
Type	ACK
Code	2.05 Content
MID	37437
Token	0x0

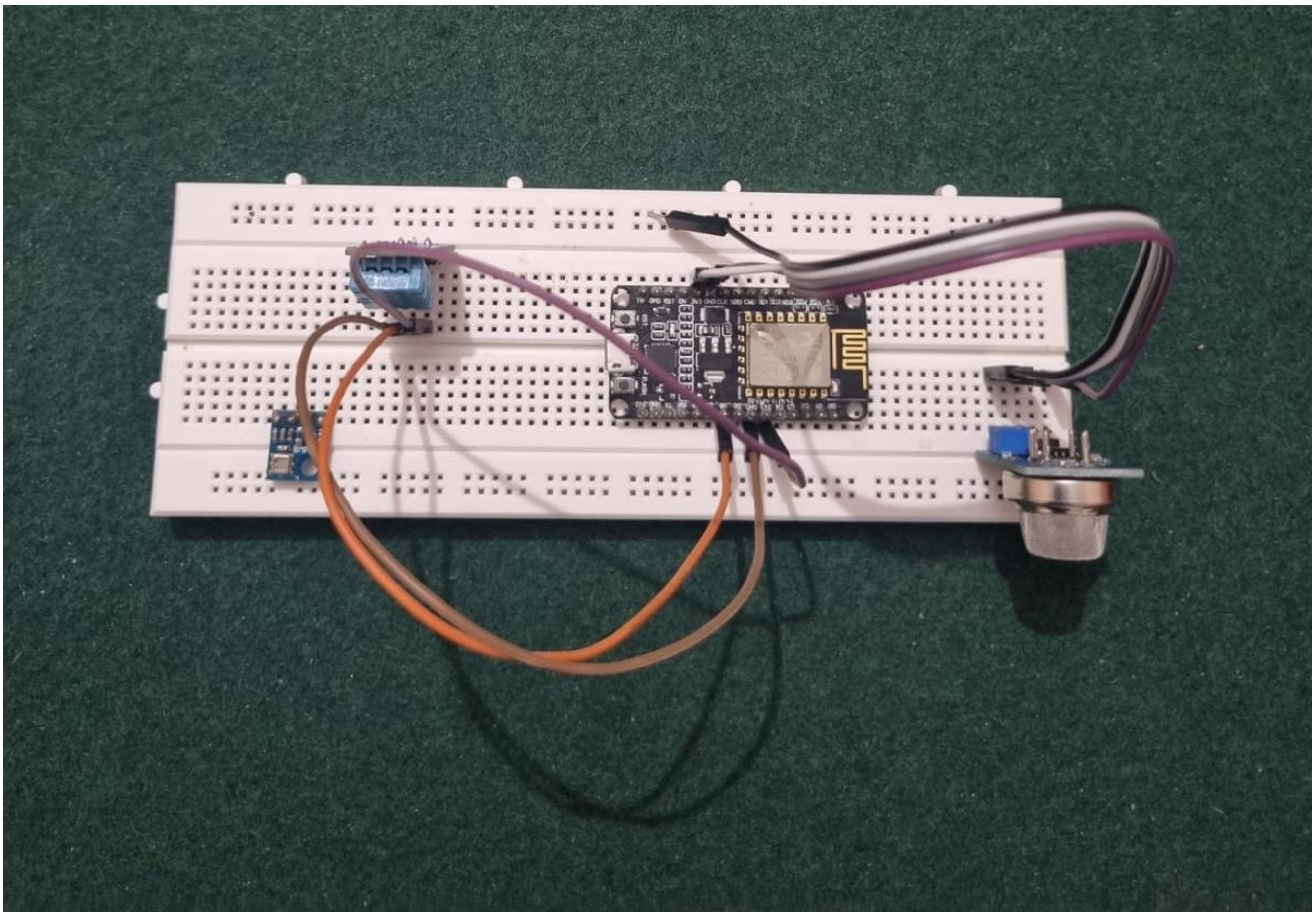
Option	Value	Raw
Content-Format (12)	0	0x000

Payload

```
28.299999
```

CoAP Message Log

```
Time      Type  Code          MID        Token  Options                              Payload
22:58:05  ACK   2.05 Content  37437      0x0    Content-Format: 0                    28.299999
22:58:05  CON   GET           37437 (0)  0x0    Uri-Path: Temperature, Block2: 0/0/64
```



APPLICATION

The weather station project using the DHT11 and MQ-135 sensors, NodeMCU board, and CoAP protocol has several potential applications, including:

- Home automation: The weather station can be used to monitor the temperature and humidity levels inside and outside the home. This information can be used to automate air conditioning and heating systems, fans, and humidifiers, creating a more comfortable indoor environment.
- Agriculture: Farmers can use the weather station to monitor the temperature and humidity levels in their fields. This information can be used to determine the optimal planting times and help farmers make informed decisions about irrigation and fertilization.
- Environmental monitoring: The weather station can be used to monitor air quality and weather conditions in urban and industrial areas. This information can be used to detect air pollution levels and take action to reduce pollution.
- Research: The weather station can be used in scientific research to monitor weather conditions and air quality in a specific location. The data collected can be used to analyze weather patterns and pollution levels over time.
- Education: The weather station can be used in schools and universities to teach students about weather and air quality monitoring. Students can learn how to collect and analyze data and use it to make informed decisions.

In conclusion, the weather station project using the DHT11 and MQ-135 sensors, NodeMCU board, and CoAP protocol has numerous potential applications in home automation, agriculture, environmental monitoring, research, and education.

CONCLUSION

In conclusion, building a weather station using the DHT11 and MQ-135 sensors, NodeMCU board, and CoAP protocol is an excellent project for anyone interested in learning more about the Internet of Things (IoT) and sensor technology. The project offers a variety of potential applications, including home automation, agriculture, environmental monitoring, research, and education.

The DHT11 sensor provides accurate temperature and humidity measurements, while the MQ-135 sensor detects a range of air pollutants. The NodeMCU board acts as the central hub, collecting sensor data and sending it over the network using the CoAP protocol.

By building this weather station, individuals can gain hands-on experience in electronics, programming, and network communication. Additionally, the weather station provides valuable data that can be used to make informed decisions about indoor and outdoor environments.

Overall, the weather station project using the DHT11 and MQ-135 sensors, NodeMCU board, and CoAP protocol is an exciting and practical project that offers many opportunities for learning and experimentation.

REFERENCE

GitHub - Alv3s/Thing.CoAP: Thing.CoAP

https://github.com/automote/ESP-CoAP

GitHub - mkovatsc/Copper4Cr: Copper (Cu) CoAP user-agent for Chrome (JavaScript implementation)

GitHub - Phoenix1747/MQ135: 📑 🌀 Arduino library for the MQ135 air quality sensor. Allows for temperature and humidity corrected readings.