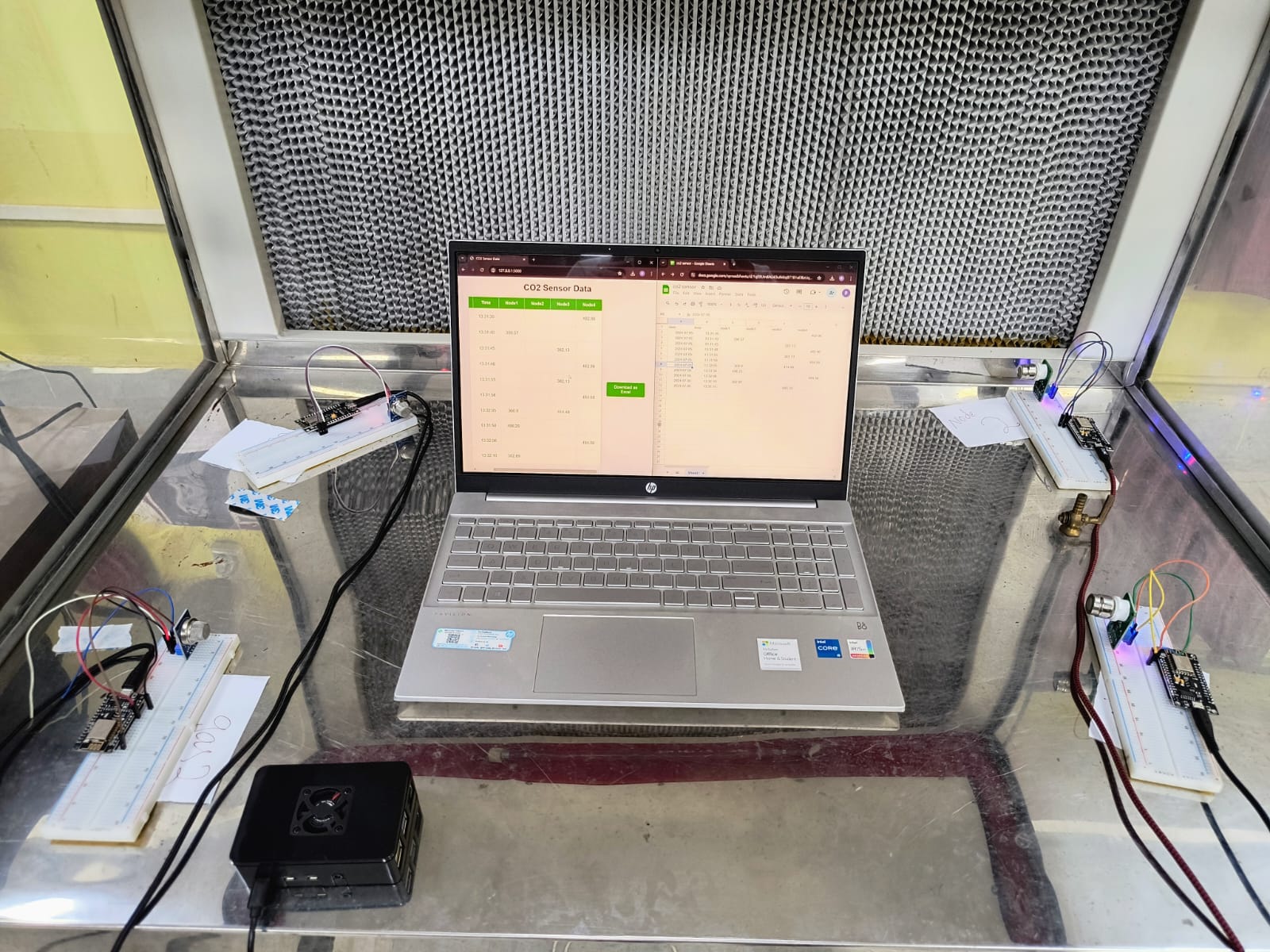
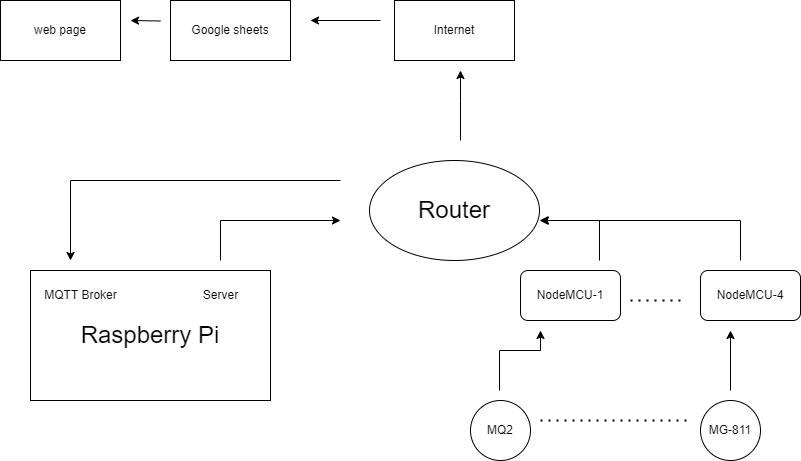
System:



Architecture :



The above monitoring system architecture is designed to provide real-time CO2 concentration data collection, processing, and visualization. The core of the system is a Raspberry Pi, functioning as both an MQTT broker and a server, facilitating communication between various components. The system includes NodeMCU devices connected to MQ2 and MG-811 sensors, which detect CO2 levels in ppm. Data from these sensors is transmitted to the Raspberry Pi via the router, which ensures network connectivity. The Raspberry Pi processes this data and makes it accessible through a web interface for real-time monitoring and logs it to Google Sheets for further analysis. In this architecture focused on monitoring CO2 concentrations, extensive testing and evaluation were conducted using both the MQ2 and MG-811 sensors. The results conclusively demonstrated that the MG-811 sensor is exceptionally accurate and reliable for CO2 level monitoring in parts per million (ppm). By leveraging the capabilities of the MG-811 sensor, the system provides dependable data that is essential for real-time decision-making and long-term trend analysis. Additionally, the system's internet connectivity allows for remote access, enabling users to monitor the environment from any location. This architecture highlights a combination of efficient data communication, cloud integration, and low-cost implementation, making it a scalable and versatile solution for Co2 levels monitoring. Also this architecture underscores the system's adaptability, affordability, and scalability, making it an ideal solution for diverse CO2 monitoring needs across various settings.

Code for NodeMCU :

#include <ESP8266WiFi.h>

#include <PubSubClient.h>

const char\* ssid = "AKCSIT407";

const char\* password = "Akcsit#407";

const char\* mqtt\_server = "192.15.2.105";  // Replace with your MQTT broker's IP address

WiFiClient espClient;

PubSubClient client(espClient);

const int gasSensorPin = A0;  // Analog pin for MQ-2 sensor

long lastMeasure = 0;

const char\* nodeID = "node4";  // Unique identifier for this NodeMCU

// Calibration variables

float zeroPointVoltage = 29;  // Adjust this with your measured zero-point voltage in mV

float voltageToCO2Factor = 3.14;  // Replace with actual conversion factor

void setup\_wifi() {

  delay(10);

  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());

}

void callback(char\* topic, byte\* message, unsigned int length) {

  Serial.print("Message arrived on topic: ");

  Serial.print(topic);

  Serial.print(". Message: ");

  for (int i = 0; i < length; i++) {

    Serial.print((char)message[i]);

  }

  Serial.println();

}

void reconnect() {

  while (!client.connected()) {

    if (client.connect(nodeID)) {  // Unique client ID for NodeMCU

      client.subscribe("/esp8266/gas4");

    } else {

      Serial.print("failed, rc=");

      Serial.print(client.state());

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

      delay(5000);

    }

  }

}

void setup() {

  Serial.begin(115200);

  setup\_wifi();

  client.setServer(mqtt\_server, 1883);

  client.setCallback(callback);

  // Perform calibration here

  // Measure and set the zeroPointVoltage based on actual measurements

  // Example:

  // zeroPointVoltage = 0.5;  // Replace with your measured zero-point voltage

  Serial.print("Zero-point voltage (mV): ");

  Serial.println(zeroPointVoltage);

}

float calculateCO2Concentration(int sensorValue) {

  float sensorVoltage = sensorValue \* (3300.0 / 1024.0);  // Convert ADC value to voltage (mV)

  float co2Concentration = abs((sensorVoltage - zeroPointVoltage) \* voltageToCO2Factor);

  return co2Concentration;

}

void loop() {

  if (!client.connected()) {

    reconnect();

  }

  client.loop();

  long now = millis();

  if (now - lastMeasure > 10000) {

    lastMeasure = now;

    int rawGasValue = analogRead(gasSensorPin);  // Raw analog reading

    float co2Concentration = calculateCO2Concentration(rawGasValue);

    static char payload[100];

    snprintf(payload, 100, "{\"nodeID\":\"%s\", \"rawValue\":%d, \"co2Concentration\":%.2f}", nodeID, rawGasValue, co2Concentration);

    // Publish to a unique topic for this node

    String topic = String("/esp8266/gas/") + nodeID;

    client.publish(topic.c\_str(), payload, true);  // Publish with QoS 1 to ensure message delivery

    // Print values to Serial Monitor

    Serial.print("Raw Gas Sensor Value: ");

    Serial.println(rawGasValue);

    Serial.print("CO2 Concentration: ");

    Serial.println(co2Concentration);

  }

  delay(random(500, 1500)); // Adding a slight random delay to avoid message collision

}

Code for Raspberry pi:

import paho.mqtt.client as mqtt

import requests

import json

import time

import threading

# MQTT settings

MQTT\_BROKER = "192.15.2.105"  # Your Raspberry Pi's IP address

MQTT\_PORT = 1883

MQTT\_TOPIC = "/esp8266/gas/#"  # Wildcard to subscribe to all nodes

# Google Sheets web app URL

WEB\_APP\_URL = "https://script.google.com/macros/s/AKfycbzKhq4l2hoO3GgXH1Ka-uUrQP5EckIipVJDxzliKwoCBs0CmPQVgrwHy\_XnDLt\_6Xvl/exec"  # Replace with your deployment URL

def on\_connect(client, userdata, flags, rc):

    print("Connected with result code " + str(rc))

    client.subscribe(MQTT\_TOPIC)

def send\_to\_google\_sheets(payload):

    try:

        response = requests.post(WEB\_APP\_URL, json=payload)

        print(f"Response from web app: {response.text}")

    except requests.RequestException as e:

        print(f"Error sending data to web app: {e}")

def on\_message(client, userdata, msg):

    try:

        # Attempt to decode the JSON payload

        data = json.loads(msg.payload.decode())

        print(f"Received data: {data}")  # Debugging: print received data

        # Check for expected keys in the payload

        if 'nodeID' in data and 'co2Concentration' in data and 'rawValue' in data:

            node\_id = data['nodeID']

            sensor\_value = data['co2Concentration']

            raw\_value = data['rawValue']

            timestamp = time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime())

            payload = {

                "timestamp": timestamp,

                "nodeID": node\_id,

                "sensorValue": sensor\_value,

                "rawValue": raw\_value

            }

            # Send data to Google Sheets in a separate thread

            threading.Thread(target=send\_to\_google\_sheets, args=(payload,)).start()

        else:

            print("Error: Expected keys 'nodeID', 'rawValue', and 'co2Concentration' not found in the payload.")

    except json.JSONDecodeError as e:

        print(f"Error: Received payload is not a valid JSON. Exception: {e}")

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

try:

    client.connect(MQTT\_BROKER, MQTT\_PORT, 60)

    client.loop\_forever()

except Exception as e:

    print(f"Error connecting to MQTT broker: {e}")

Google Apps script code:

function doPost(e) {

  var sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

  var data;

  try {

    data = JSON.parse(e.postData.contents);

  } catch (f) {

    return ContentService

      .createTextOutput(JSON.stringify({"status": "error", "message": f.message}))

      .setMimeType(ContentService.MimeType.JSON);

  }

  var timestamp = new Date();  // Get current date and time

  var date = Utilities.formatDate(timestamp, Session.getScriptTimeZone(), "yyyy-MM-dd");

  var time = Utilities.formatDate(timestamp, Session.getScriptTimeZone(), "HH:mm:ss");

  // Print or log the date and time to verify

  console.log("Date:", date);

  console.log("Time:", time);

  var nodeID = data.nodeID;

  var sensorValue = data.sensorValue;

  // Determine the column based on nodeID

  var columnMap = {

    "node1": 3,  // Adjust column numbers based on your specific sheet layout

    "node2": 4,

    "node3": 5,

    "node4": 6

    // Add more nodes as needed

  };

  var column = columnMap[nodeID];

  if (!column) {

    return ContentService

      .createTextOutput(JSON.stringify({"status": "error", "message": "Invalid nodeID"}))

      .setMimeType(ContentService.MimeType.JSON);

  }

  // Find the last row in the sheet

  var lastRow = sheet.getLastRow() + 1;

  // Insert the date, time, and sensor value into the appropriate columns

  sheet.getRange(lastRow, 1).setValue(date);  // Date in the first column

  sheet.getRange(lastRow, 2).setValue(time);  // Time in the second column

  sheet.getRange(lastRow, column).setValue(sensorValue);  // Sensor value in the appropriate column

  return ContentService

    .createTextOutput(JSON.stringify({"status": "success"}))

    .setMimeType(ContentService.MimeType.JSON);

}

Flask App Code:

from flask import Flask, jsonify, render\_template

import gspread

from oauth2client.service\_account import ServiceAccountCredentials

app = Flask(\_\_name\_\_)

# Define the scope and credentials

scope = ['https://spreadsheets.google.com/feeds',

         'https://www.googleapis.com/auth/drive']

creds = ServiceAccountCredentials.from\_json\_keyfile\_name(r"C:\Users\vscgo\OneDrive\Desktop\scriptapp\co2-sensor-428405-2e5755728d86.json", scope)

client = gspread.authorize(creds)

# Define the Google Sheet

sheet = client.open("co2 sensor").sheet1  # Replace with your actual sheet name

@app.route('/')

def index():

    return render\_template('index.html')

@app.route('/sensor-data', methods=['GET'])

def get\_sensor\_data():

    # Define the expected headers based on your sheet

    expected\_headers = ["date", "time", "node1", "node2", "node3", "node4"]  # Update these as per your actual headers

    # Fetch all records using expected headers

    data = sheet.get\_all\_records(expected\_headers=expected\_headers)

    return jsonify(data)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

Web Page Code :

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>CO2 Sensor Data</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            margin: 0;

            padding: 0;

            background-color: #f4f4f9;

            display: flex;

            flex-direction: column;

            align-items: center;

            height: 100vh;

            overflow: hidden; /\* Prevent body scrolling \*/

        }

        h1 {

            text-align: center;

            color: #333;

            margin: 20px 0;

        }

        .content {

            display: flex;

            width: 90%;

            height: calc(100vh - 80px); /\* Adjust for header and margin \*/

        }

        .table-container {

            flex-grow: 1; /\* Take up remaining space \*/

            overflow: auto; /\* Enable both vertical and horizontal scrolling \*/

            border: 1px solid #ddd;

            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

            margin-right: 20px; /\* Space between table and button \*/

        }

        table {

            width: 100%;

            border-collapse: collapse;

            min-width: 600px; /\* Ensure the table is wide enough for horizontal scrolling \*/

        }

        th, td {

            padding: 12px;

            border: 1px solid #ddd;

            text-align: center;

            width: 16.66%; /\* Ensure each column takes up an equal width \*/

        }

        th {

            background-color: #4CAF50;

            color: white;

            position: sticky;

            top: 0;

            z-index: 1;

        }

        td {

            background-color: #fff;

        }

        .button-container {

            display: flex;

            align-items: center;

        }

        button {

            padding: 10px 20px;

            background-color: #4CAF50;

            color: white;

            border: none;

            cursor: pointer;

            border-radius: 5px;

            font-size: 16px;

        }

        button:hover {

            background-color: #45a049;

        }

    </style>

    <!-- Include the SheetJS library -->

    <script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.17.0/xlsx.full.min.js"></script>

    <script>

        async function fetchSensorData() {

            try {

                const response = await fetch('/sensor-data');

                const data = await response.json();

                // Clear previous data

                const tbody = document.querySelector('tbody');

                tbody.innerHTML = '';

                data.forEach(row => {

                    const newRow = document.createElement('tr');

                    newRow.innerHTML = `

                        <td>${row.date}</td>

                        <td>${row.time}</td>

                        <td>${row.node1}</td>

                        <td>${row.node2}</td>

                        <td>${row.node3}</td>

                        <td>${row.node4}</td>

                    `;

                    tbody.appendChild(newRow);

                });

            } catch (error) {

                console.error('Error fetching sensor data:', error);

            }

        }

        function downloadExcel() {

            const table = document.querySelector('table');

            const wb = XLSX.utils.table\_to\_book(table, {sheet: "Sheet1"});

            XLSX.writeFile(wb, 'CO2\_Sensor\_Data.xlsx');

        }

        setInterval(fetchSensorData, 5000);  // Update every 5 seconds

        document.addEventListener('DOMContentLoaded', fetchSensorData);  // Initial load

    </script>

</head>

<body>

    <h1>CO2 Sensor Data</h1>

    <div class="content">

        <div class="table-container">

            <table>

                <thead>

                    <tr>

                        <th>Date</th>

                        <th>Time</th>

                        <th>Node1</th>

                        <th>Node2</th>

                        <th>Node3</th>

                        <th>Node4</th>

                    </tr>

                </thead>

                <tbody>

                    <!-- Data rows will be inserted here by JavaScript -->

                </tbody>

            </table>

        </div>

        <div class="button-container">

            <button onclick="downloadExcel()">Download as Excel</button>

        </div>

    </div>

</body>

</html>