Air Quality Level with ESP8266 and Environmental Sensors (AQLES)

Table of Contents

1.0 Introduction

This project, titled "Air Quality Level with ESP8266 and Environmental Sensors", AQLES aims to provide real-time air quality monitoring, leveraging the capabilities of IoT technology to create an environmental monitoring system. The initiative focuses on developing a solution to track and report air quality levels, thereby offering valuable insights into the surrounding environmental conditions. Central to this project is the a microcontroller compatible with Arduino, which serves as the backbone of the system, enabling communication and data processing. Alongside the microcontroller, the project employs a multi-sensor for temperature and humidity measurements and the gas sensor to detect gas concentration, particularly CO2, in the atmosphere. These sensors collectively ensure comprehensive monitoring of critical environmental parameters, contributing to the system's accuracy and effectiveness. Hosted on a live domain, the system aims to deliver insights into environmental conditions, making it a tool for both personal and public use. By continuously monitoring and reporting real-time data, this project not only highlights the potential of IoT in environmental applications but also underscores the importance of proactive air quality management in fostering healthier living environments. Additionally, the system integrates a Telegram bot to provide users with instant access to the latest readings and to alert them if there is a significant drop in gas concentration, which enables the users to stay informed and respond promptly to changes in air quality through their mobile devices.

1.1 Objectives

The primary goal of this project is to establish a robust system capable of:

- **Continuous Monitoring**: Monitoring temperature, humidity, and C02 concentration as parameters in real-time, then a corrected CO2 concentration is generated.
- **Real-time Data Management**: Collecting sensor data, processing it on a microcontroller, and transmitting information to the AwardSpace-hosted server.

- **Alert System**: Issuing timely alerts or notifications based on predefined thresholds for air quality metrics, ensuring prompt responses to changing environmental conditions.

2.0 System Design and Architecture

The collected data is processed locally on the ESP8266, leveraging onboard algorithms to compute air quality indices. Subsequently, the processed data is securely transmitted over WiFi to a server hosted on AwardSpace. This server acts as the backend, storing and managing the influx of sensor data, and providing users with accessible web interfaces to monitor real-time readings, analyze historical trends, and receive alerts based on predefined thresholds.

2.1 Hardware Components

Below are the specific components utilized for the project.

- ESP8266 NodeMCU V2

Figure 1: ESP8266 NodeMCU V2

The ESP8266 supports versatile programming options, including Arduino IDE, which simplifies development for sensor integration and data processing.

- DHT22 Temperature and Humidity Sensor

Figure 2: DHT22 with Board

Connected directly to the ESP8266, the DHT22 enables continuous monitoring of ambient temperature and humidity levels, contributing data points to the overall air quality assessment.

- MQ135 Gas Sensor

Figure 3: MQ135 Gas Sensor with Board

The MQ135 gas sensor detects a wide range of gases present in the atmosphere, including CO2, ammonia, benzene, and alcohol. It operates on the principle of conductivity changes in the presence of specific gases, providing semi-quantitative measurements suitable for air quality monitoring applications. Note that this sensor needs a 24-hour pre-heating before using it for the first time. It is a one-time process. Then, to get a stable value, leave it for another 3 to 5 minutes.

2.2 Data Storage

The system utilizes phpMyAdmin, a popular web-based interface for managing MySQL databases, hosted on AwardSpace.

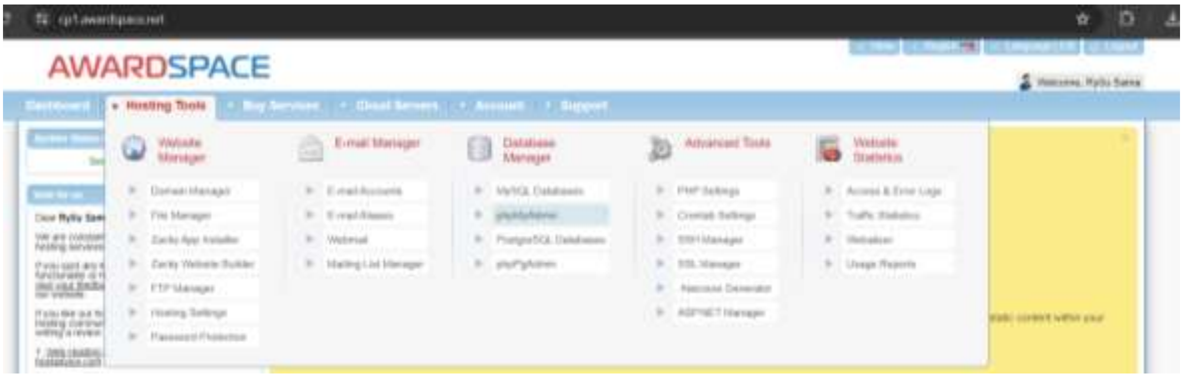


Figure 4: AwardSpace Database Manager Options

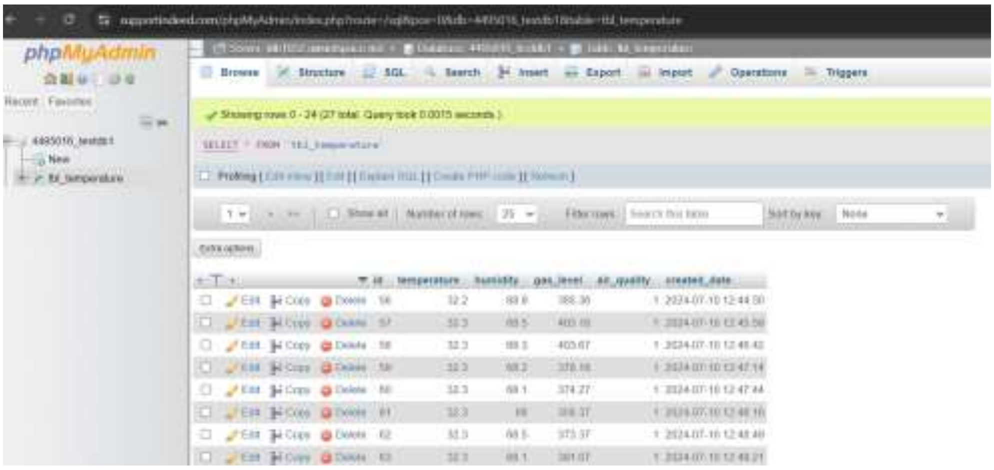


Figure 5: phpMyAdmin Database Manager

Data from sensors is structured into a defined schema within the MySQL database managed through phpMyAdmin. Each sensor reading is typically associated with timestamps.

2.3 Software Application

Arduino IDE is used as the base of the project, to retrieve the readings from the sensors. The software can also be used to send data to the social media platforms. In this case, Telegram is the receiver.

Figure 6: Arduino IDE

Figure 7: Telegram

2.4 Block Diagram

The data are retrieved from MQ135 and DHT22 Sensors, with Arduino as the medium to connect to AwardSpace and Telegram.

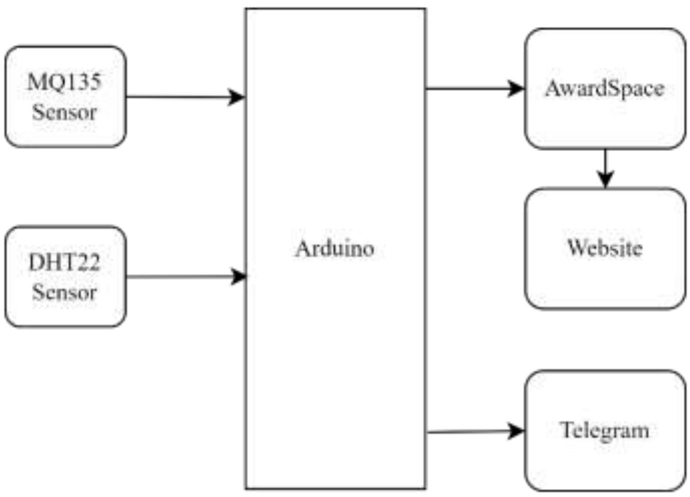


Figure 8: Block Diagram for AQLES

2.5 Data Flow Diagram

The processing steps are as follows:

- Arduino attempt to connect to the server.
- Config file for the domain and database is received.
- Receive data from sensors on their respective pins.
- Air Quality is classified.
- Data can be sent to Telegram bot, where appropriate command will print the readings from the sensors. Telegram bot alerts sudden drop in C02 concentration. Then the process ends.
- On the other hand, the data will also be sent to the server. There, latest readings will be displayed, followed by recent 30 readings, the trend chart, and the average of all the readings. Then, the process ends.

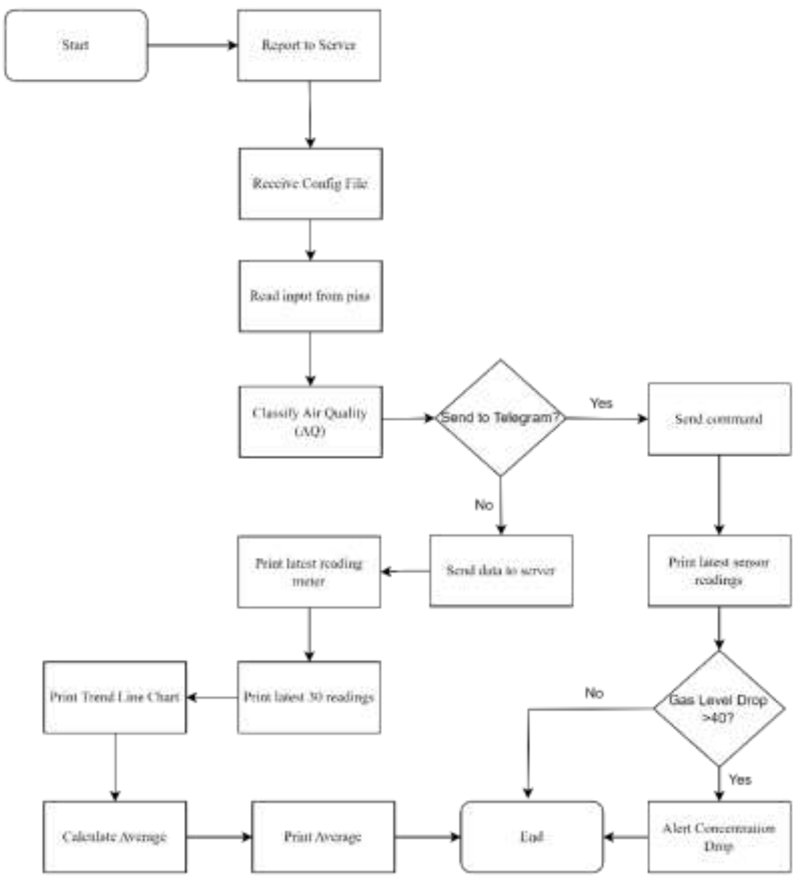


Figure 9: Data Flow Diagram for AQLES

2.6 Fritzing Schematics

The DHT22 and MQ135 to be utilized operates through the I2C communication protocol, requiring a connection to the appropriate I2C pins on ESP8266. Using an ESP8266, these pins are configured:

- D3: DHT22 Data
- A0: MQ135 Data

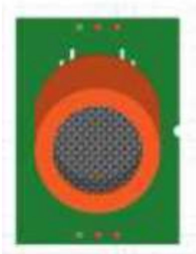


Figure 10: ESP8266 NodeMCUV2 **Figure 11: ESP8266 NodeMCUV2**

Figure 12: ESP8266 NodeMCUV2

Which results to the following Breadboard Schematic from Fritzing:

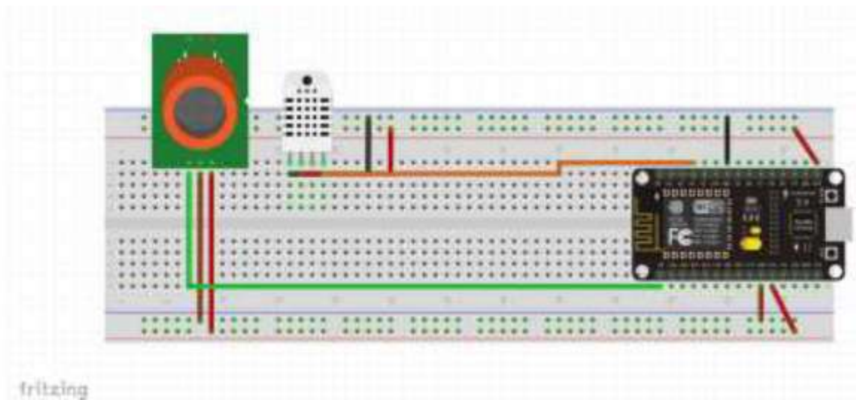


Figure 13: ESP8266 Connected to DHT22 and MQ135

3.0 Implementation

The database design will be as follows:

Table 1: Database Table

```
CREATE TABLE `tbl_temperature` (

        `id` int(11) NOT NULL,
        `temperature` float NOT NULL,
        `humidity` float NOT NULL,
        `gas_level` float NOT NULL,
        `air_quality` int(11) NOT NULL,
        `created_date` timestamp NOT NULL DEFAULT
      CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The temperature is classified from too cold, cold, normal and hot. The range is based on the climate context of Malaysia for the current climatology, 1991-2020, derived from observed, historical data (Climatic Research Unit (CRU) of University of East Anglia, n.d.). The data chosen as the range is on the month of July.

Table 2: Temperature Category

Class	Range (in Celsius)
Too Cold	<22.29
Cold	22.29<=x<26.51
Normal	26.51<=x<30.79
Hot	x>=30.79

On the other hand, relative humidity is recorded in range of values following the climate for the area northwest of the Peninsula (Alor Star) where the mean relative humidity varies from 72% to 87% (Malaysian Meteorological Department, n.d.).

Table 3: Humidity Category

Class	Range (in percentage)

Low	x<72
Normal	72<=x<87
Hight	x>=87
Hot	x>=30.79

3.1 Web Page

The web page will be split into several sections, which are:

- Temperature and Humidity Meter

  This section will reveal the latest readings of temperature and humidity from the DHT22 sensor. The meter is from the Google Charts library (Google, n.d) using scripts, in index.php, with the method drawTemperatureChart() and drawHumidityChart() respectively.

Figure 14: Temperature and Humidity Meter

- Recent Readings Table

  This section reveals the latest 30 readings from both DHT22 and MQ135 sensors, and the classified value of the air quality level (AQ), including the timestamps for each reading, from the tbl_temperature table. The method is in the getdata.php, which is:

  $sql = "SELECT * FROM tbl_temperature ORDER BY id DESC LIMIT 30";

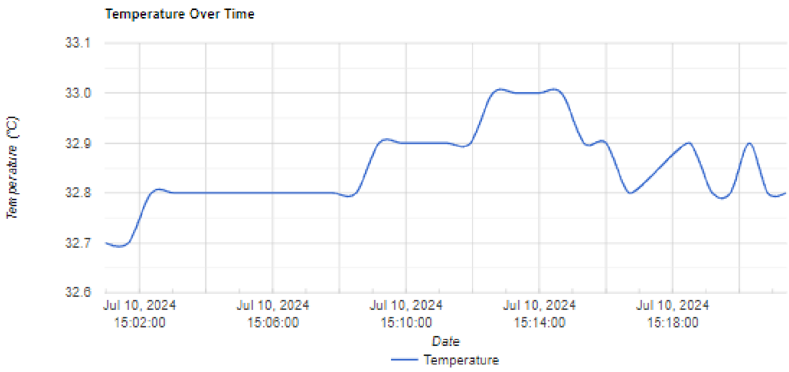
#	Temperature (°C)	Humidity (%)	CO2 Concentration (PPM)	AQ Level	Date Time
1	32.8	70.4	[illegible]	1	[illegible]
2	32.8	70.3	[illegible]	1	[illegible]
3	32.9	70.5	[illegible]	1	[illegible]
4	32.8	70.4	[illegible]	1	[illegible]
5	32.8	70.1	[illegible]	1	[illegible]
6	32.9	70.8	[illegible]	1	[illegible]
7	32.8	70.6	[illegible]	1	[illegible]
8	32.8	70.4	[illegible]	1	[illegible]
9	32.9	70.7	[illegible]	1	[illegible]
10	33	70.1	[illegible]	1	[illegible]

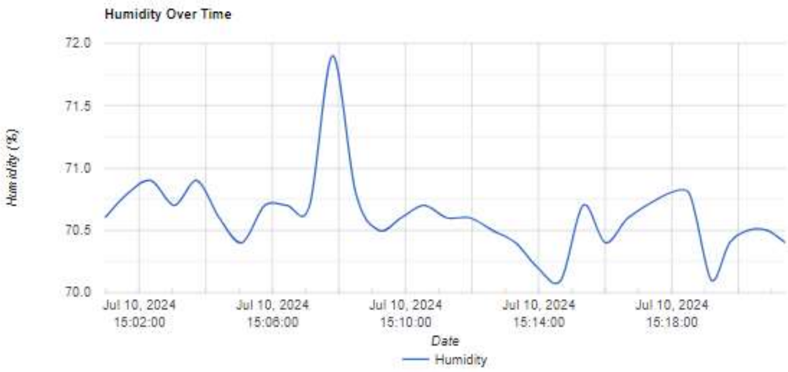
Figure 15: tbl_temperature Table

- Trend Line Chart

  For all the readings from the sensors, line charts are displayed to check on the trends for the readings, set a pivot on x-axis for each 4 minutes. The charts use the Google Charts library (Google, n.d) using script in index.php, and method drawLineCharts(). Data collected with the line in getdata.php:

```
while ($row = $result->fetch_assoc()) {
   $data[] = $row;
}
```





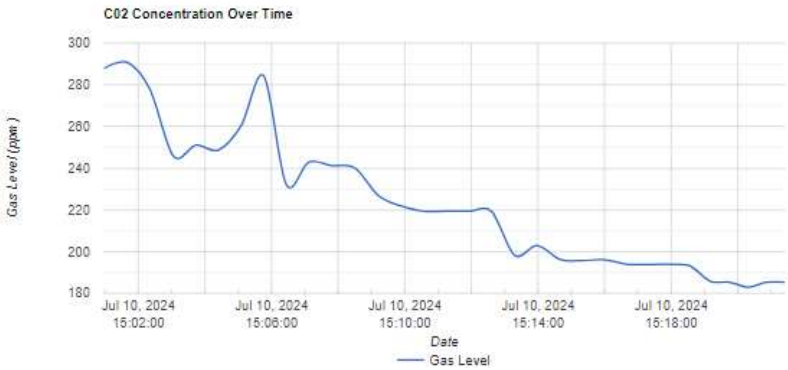


Figure 16: Line Chart for Humidity, C02

- Average Readings

  The total average from all the readings is displayed.

Figure 17: Line Chart for Temperature, Humidity, C02

The data is collected using these lines in getdata.php:

Table 4: Calculation and Retrieval of Averages

```
while ($row = $result->fetch_assoc()) {
    $data[] = $row;
    $totalTemp += $row['temperature'];
    $totalHumidity += $row['humidity'];
    $totalGas += $row['gas_level'];
    $count++;
}

$averages = [
```

```
    'avg_temp' => $count ? $totalTemp / $count : 0,

    'avg_humidity' => $count ? $totalHumidity / $count : 0,

    'avg_gas' => $count ? $totalGas / $count : 0

];



$response = [

    'data' => $data,

    'averages' => $averages];
```

3.2 Telegram

A Telegram bot is created by another telegram bot, named TheBotFather. From there, AQRead bot is created, with predefined commands on the Arduino IDE code.

Figure 18: TheBotFather and AQRead bots for Telegram

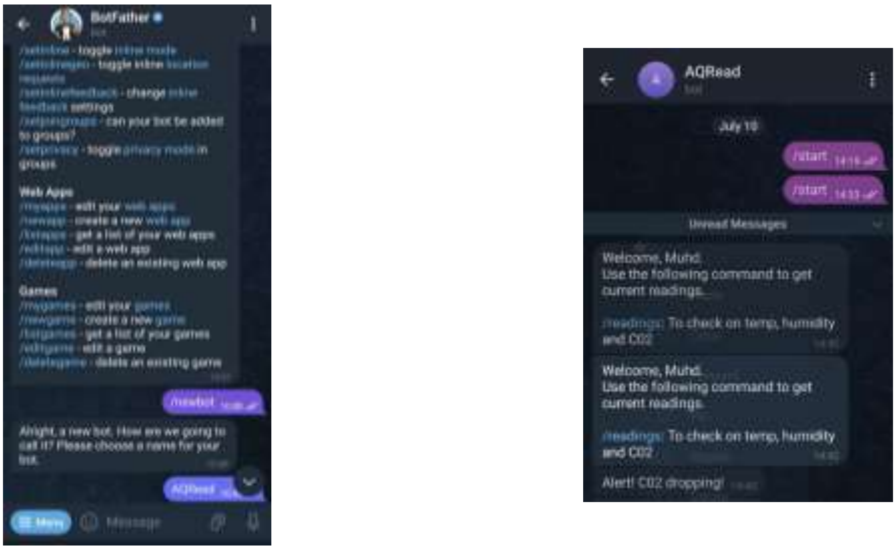


Figure 19: TheBotFather and AQRead bots for Telegram

From there, AQRead bot is created, with predefined commands on the Arduino IDE code. The BotFather is used to get the token API and user ID to use in the Arduino code later. ID is retrieved from IDBot with the command /getID.

Before running the Arduino code, certain values in the file MQ135.h in the Arduino library folder needs to be adjusted which are RZERO and ATMOC02. Run getCorrectedRZero() to get get the resistance RZero of the sensor for calibration purposes (Cesanta Software Ltd., n.d.). ATMOC02 values is changed according to global climate data (Milman, 2024).

Figure 20: MQT135.h File

3.3 AwardSpace

The web hosting used is a free web hosting provider, AwardSpace. The free plan will allow for a single database, and up to two free domains. Proceed create a new domain by clicking on Hosting Tools > Domain Manager > Create A free sub domain.

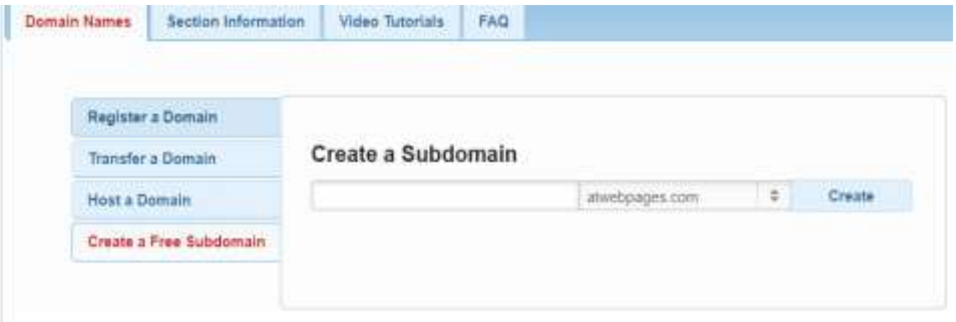


Figure 21: Create a Domain

Then, go to Hosting Tools > MySQL Databases and create a new database.

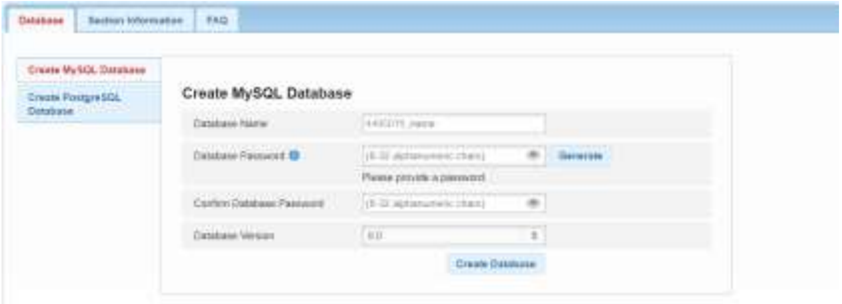


Figure 22: Create a Database

Next, go to Hosting Tools > File Manager, select the folder of newly created domain to upload all the php and assets files into the AwardSpace.

Figure 23: AwardSpace File Manager

4.0 Testing and Validation Results

The testing and validation are conducted in two parts. The first part is to check whether all the readings are saved and visualized correctly.

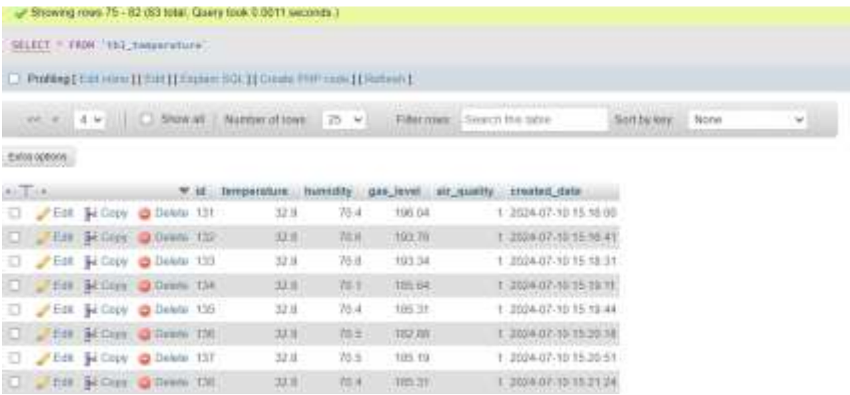


Figure 24: Latest Row of Data in phpMyAdmin

Figure 25: Latest Readings of Temperature and Humidity for Meter

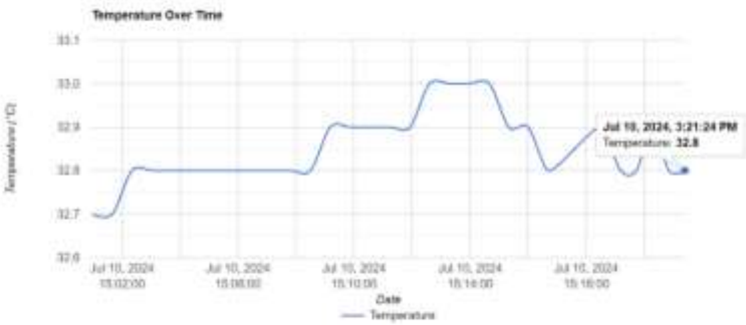


Figure 26: Latest Readings of Temperature in Line Chart

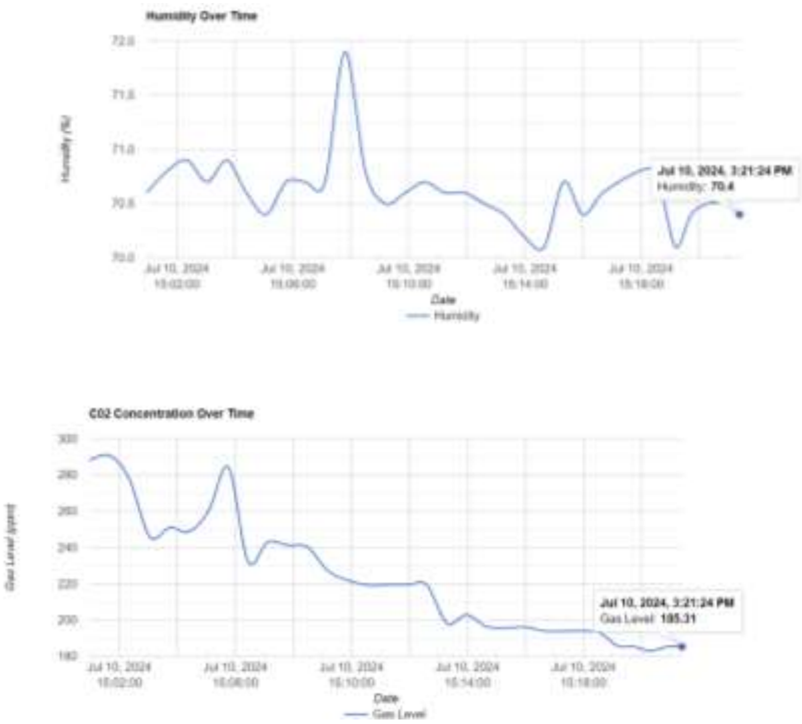


Figure 27: Latest Readings of Humidity and C02 in Line Chart

The Figure 24, 25, 26, and 27 proves that the line chart and meter work fine, since the data in latest row of the table in database tallies with the data point visualized in the meter and the line chart. This also means that each row of data is recorded smoothly, which results in proper average results for the three data (temperature, humidity and C02).

The second part is for the Telegram usage. The command /readings were tested, and to validate, check if recent data in the database is the same as the one given by the AQRead bot.

Figure 28: Readings of Temperature, Humidity and C02 from Telegram

In Figure 28, the readings from /readings are not stored in the database, since it uses current data while the ones in the table are stored by intervals. However, the validation still can be done by comparing the rows with the timestamps close to the time of the command being executed.

☐	Edit	Copy	Delete	150	32.2	70.8	210.96	1	2024-07-10 17:06:33
☐	Edit	Copy	Delete	151	32.2	71.1	159.74	1	2024-07-10 17:07:08

Figure 26: Readings in phpMyAdmin

As seen in Figure 2, the recent two data have difference by more than 40, for gas level column. This shows that the level of the C02 is indeed dropping, as notified by the AQRead bot.

5.0 Conclusion

In conclusion, the AQLES successfully demonstrated the integration of ESP8266, DHT22, and MQ135 sensors to collect and analyze environmental data. The system provides real-time monitoring of temperature, humidity, and CO_2 levels, ensuring timely alerts and accurate assessments of air quality.

Key achievements of the project include:

- **Real-time Data Collection**: The system continuously gathers and updates environmental data, providing users with up-to-date information on air quality.
- **Web-based Monitoring**: Using PHP and Google Charts, the data is effectively visualized on a web interface, making it accessible and comprehensible for users.
- **Automated Alerts**: The integration with Telegram via the UniversalTelegramBot library ensures that users receive immediate notifications when significant changes in CO_2 levels are detected, particularly focusing on drops in CO_2 concentration, which can indicate potential issues.

5.1 Future Work

Building upon the successful implementation of the air quality monitoring system, several avenues for future work can be pursued to enhance the system's capabilities and expand its applications:

- **Expanded Sensor Network**: Deploying additional sensors across a wider geographical area would provide more comprehensive coverage and detailed insights into regional air quality variations. This could involve both fixed and mobile sensor units.

- **Integration with Other Environmental Sensors**: Adding sensors for monitoring other pollutants such as particulate matter (PM2.5 and PM10), nitrogen dioxide (NO_2), and sulfur dioxide (SO_2) can offer a more complete picture of air quality and environmental health.

- **Advanced Data Analytics**: Implementing machine learning algorithms for predictive analysis can provide early warnings about deteriorating air quality. These models could analyze historical data to predict future trends and identify potential hazards before they become critical.

- **Improved Alert System**: Enhancing the alert system to provide more detailed notifications, including possible health advice and actions to mitigate exposure to poor air quality. Integration with other communication platforms (e.g., SMS, email) can broaden the reach of these alerts.

- **User Interface Enhancements**: Improving the web interface to include more interactive features, such as real-time data filtering, customizable dashboards, and detailed historical data analysis. Mobile application development could also be considered for more accessible monitoring on-the-go.

References

Cesanta Software Ltd. (n.d.). *Mongoose OS documentation*. https://mongooseos.com/docs/mongoose-os/api/drivers/mq135.md

Climatic Research Unit (CRU) of University of East Anglia. (n.d.). *World Bank Climate Change Knowledge Portal*. https://climateknowledgeportal.worldbank.org/country/malaysia/climate-data-historical#:~:text=Malaysia%20has%20a%20tropical%20climate,25.9%C2%B0C%20in%20May

Google for Developers. *Load the libraries*. (n.d.). https://developers.google.com/chart/interactive/docs/basic_load_libs

Malaysian Meteorological Department. (n.d.). *Laman Web Rasmi Jabatan Meteorologi Malaysia*. https://www.met.gov.my/en/pendidikan/iklimmalaysia/#Relative%20humidity

Milman, O. (2024, May 9). Record-breaking increase in CO2 levels in world's atmosphere. *The Guardian*. https://www.theguardian.com/environment/article/2024/may/09/carbon-dioxide-atmosphere-record

Code

config.php

```
<?php
define('DB_HOST', 'fdb1032.awardspace.net');
define('DB_USERNAME', '4495016_testdb1');
define('DB_PASSWORD', 'skih_3113');
define('DB_NAME', '4495016_testdb1');

define('POST_DATA_URL', 'http://samaq.atwebpages.com/sensordata.php');

//PROJECT_API_KEY is the exact duplicate of, PROJECT_API_KEY in NodeMCU
sketch file
//Both values must be same
define('PROJECT_API_KEY', 'tempQuality');



//set time zone for your country
date_default_timezone_set("Asia/Kuala_Lumpur");

// Connect with the database
$db = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD, DB_NAME);

// Display error if failed to connect
if ($db->connect_errno) {
   echo "Connection to database is failed: ".$db->connect_error;
   exit();
}
```

getdata.php

```
<?php
require 'config.php';

$sql = "SELECT * FROM tbl_temperature ORDER BY id DESC LIMIT 30";
$result = $db->query($sql);

if (!$result) {
    echo json_encode(['error' => $db->error]);
    exit();
}

$data = [];
$totalTemp = 0;
$totalHumidity = 0;
$totalGas = 0;
$count = 0;

while ($row = $result->fetch_assoc()) {
    $data[] = $row;
    $totalTemp += $row['temperature'];
    $totalHumidity += $row['humidity'];
    $totalGas += $row['gas_level'];
    $count++;
}

$averages = [
    'avg_temp' => $count ? $totalTemp / $count : 0,
    'avg_humidity' => $count ? $totalHumidity / $count : 0,
    'avg_gas' => $count ? $totalGas / $count : 0
];

$response = [
```

```
    'data' => $data,
    'averages' => $averages
];

echo json_encode($response);
?>
```

sensordata.php

```
<?php
require 'config.php';

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $api_key = escape_data($_POST["api_key"]);

    if ($api_key == PROJECT_API_KEY) {
        $temperature = escape_data($_POST["temperature"]);
        $humidity = escape_data($_POST["humidity"]);
        $gas_level = escape_data($_POST["gas_level"]); // New field for gas level
        $air_quality = escape_data($_POST["air_quality"]); // New field for air quality

        $sql = "INSERT INTO tbl_temperature (temperature, humidity, gas_level, air_quality,
created_date)
                VALUES ('$temperature', '$humidity', '$gas_level', '$air_quality', '".date("Y-m-d
H:i:s")."')";

        if ($db->query($sql) === FALSE) {
            echo "Error: " . $sql . "<br>" . $db->error;
        } else {
            echo "OK. INSERT ID: " . $db->insert_id;
        }
    } else {
        echo "Wrong API Key";
```

```
    }
} else {
    echo "No HTTP POST request found";
}

function escape_data($data) {
    global $db;
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $db->real_escape_string($data);
}
?>
```

index.php

```
<?php
require 'config.php';

// Fetch the data
$sql = "SELECT * FROM tbl_temperature ORDER BY id DESC LIMIT 30";
$result = $db->query($sql);

if (!$result) {
    die("Error: " . $sql . "<br>" . $db->error);
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <title>SAM's Air Quality Monitor</title>
    <meta charset="utf-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/css/bootstrap.min.css">
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.1/dist/js/bootstrap.bundle.min.js"></script
>
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <style>
        .chart {
            width: 100%;
            min-height: 450px;
        }
        .row {
            margin: 0 !important;
        }
        .average-section {
            text-align: center;
            margin: 20px 0;
        }
        .average-section img {
            width: 50px;
            height: 50px;
        }
    .mid {
                    text-align: center;
                    border: 7px solid green;
            }
    </style>
</head>
<body>
```

```html
<div class="container">
   <div class="mid">
         <h2>AQ Level</h2>
         <ul>
                              <li>1: Good</li>
                              <li>2: Poor</li>
                              <li>3: Dangerous</li>
                         </ul>
   </div>
   <div class="row">
      <div class="col-md-12 text-center">
         <h1>Air Quality Monitor</h1>
      </div>
      <div class="clearfix"></div>
      <div class="col-md-6">
         <div id="chart_temperature" class="chart"></div>
      </div>
      <div class="col-md-6">
         <div id="chart_humidity" class="chart"></div>
      </div>
   </div>

   <div class="row">
      <div class="col-md-12">
         <table class="table">
            <thead>
            <tr>
               <th scope="col">#</th>
               <th scope="col">Temperature (°C)</th>
               <th scope="col">Humidity (%)</th>
               <th scope="col">C02 Concentration (PPM)</th>
               <th scope="col">AQ Level</th>
```

```
                <th scope="col">Date Time</th>
            </tr>
            </thead>
            <tbody>
            <?php $i = 1; while ($row = $result->fetch_assoc()) {?>
                <tr>
                    <th scope="row"><?php echo $i++;?></th>
                    <td><?php echo $row['temperature'];?></td>
                    <td><?php echo $row['humidity'];?></td>
                    <td><?php echo $row['gas_level'];?></td>
                    <td><?php echo $row['air_quality'];?></td>
                    <td><?php echo date("Y-m-d h:i:sa ",
strtotime($row['created_date']));?></td>
                </tr>
            <?php } ?>
            </tbody>
        </table>
    </div>
</div>

<div class="row">
    <div class="col-md-12">
        <div id="line_chart_temperature" class="chart"></div>
        <div id="line_chart_humidity" class="chart"></div>
        <div id="line_chart_gas" class="chart"></div>
    </div>
</div>

<div class="mid">
        <h2>Average</h2>
</div>

<div class="row average-section">
```

```
        <div class="col-md-4">
            <img src="assets/img/therm.png" alt="Temperature">
            <h3>Temperature : <span id="avg_temp">0</span> °C</h3>
        </div>
        <div class="col-md-4">
            <img src="assets/img/hum.png" alt="Humidity">
            <h3>Humidity : <span id="avg_humidity">0</span> %</h3>
        </div>
        <div class="col-md-4">
            <img src="assets/img/gas.png" alt="Gas Level">
            <h3>C02 Concentration : <span id="avg_gas">0</span> ppm</h3>
        </div>
    </div>
</div>

<script>
    google.charts.load('current', {'packages':['gauge', 'corechart']});
    google.charts.setOnLoadCallback(drawTemperatureChart);
    google.charts.setOnLoadCallback(drawHumidityChart);
    google.charts.setOnLoadCallback(drawLineCharts);

    function drawTemperatureChart() {
        var data = google.visualization.arrayToDataTable([
            ['Label', 'Value'],
            ['Temperature', 0],
        ]);

        var options = {
            width: 1600,
            height: 480,
            redFrom: 30,
            redTo: 100,
            yellowFrom: 26,
```

```
        yellowTo: 30,
        greenFrom: 0,
        greenTo: 26,
        minorTicks: 5
      };

      var chart = new
google.visualization.Gauge(document.getElementById('chart_temperature'));
      chart.draw(data, options);

      function refreshData() {
        $.ajax({
          url: 'getdata.php',
          dataType: 'json',
          success: function (response) {
            var temperature = parseFloat(response.data[0].temperature).toFixed(2);
            var data = google.visualization.arrayToDataTable([
              ['Label', 'Value'],
              ['Temperature', eval(temperature)],
            ]);
            chart.draw(data, options);
          },
          error: function(jqXHR, textStatus, errorThrown) {
            console.log(errorThrown + ': ' + textStatus);
          }
        });
      }

      setInterval(refreshData, 1000);
    }

    function drawHumidityChart() {
      var data = google.visualization.arrayToDataTable([
```

```
    ['Label', 'Value'],
    ['Humidity', 0],
  ]);

  var options = {
    width: 1600,
    height: 480,
    redFrom: 72,
    redTo: 100,
    yellowFrom: 72,
    yellowTo: 87,
    greenFrom: 0,
    greenTo: 72,
    minorTicks: 5
  };

  var chart = new
google.visualization.Gauge(document.getElementById('chart_humidity'));
  chart.draw(data, options);

  function refreshData() {
    $.ajax({
      url: 'getdata.php',
      dataType: 'json',
      success: function (response) {
        var humidity = parseFloat(response.data[0].humidity).toFixed(2);
        var data = google.visualization.arrayToDataTable([
          ['Label', 'Value'],
          ['Humidity', eval(humidity)],
        ]);
        chart.draw(data, options);
      },
      error: function(jqXHR, textStatus, errorThrown) {
```

```
                console.log(errorThrown + ': ' + textStatus);
            }
        });
    }

    setInterval(refreshData, 1000);
}

function drawLineCharts() {
    function refreshLineChartData() {
        $.ajax({
            url: 'getdata.php',
            dataType: 'json',
            success: function (response) {
                var temperatureData = [['Date', 'Temperature']];
                var humidityData = [['Date', 'Humidity']];
                var gasData = [['Date', 'Gas Level']];

                response.data.reverse().forEach(function (row) {
                    var date = new Date(row.created_date);
                    temperatureData.push([date, parseFloat(row.temperature)]);
                    humidityData.push([date, parseFloat(row.humidity)]);
                    gasData.push([date, parseFloat(row.gas_level)]);
                });

                drawLineChart('line_chart_temperature', temperatureData, 'Temperature Over
Time', 'Temperature (°C)');
                drawLineChart('line_chart_humidity', humidityData, 'Humidity Over Time',
'Humidity (%)');
                drawLineChart('line_chart_gas', gasData, 'C02 Concentration Over Time', 'Gas
Level (ppm)');

                // Update averages
```

```
                $('#avg_temp').text(parseFloat(response.averages.avg_temp).toFixed(2));

$('#avg_humidity').text(parseFloat(response.averages.avg_humidity).toFixed(2));
                $('#avg_gas').text(parseFloat(response.averages.avg_gas).toFixed(2));
            },
            error: function(jqXHR, textStatus, errorThrown) {
                console.log(errorThrown + ': ' + textStatus);
            }
        });
    }

    function drawLineChart(elementId, data, title, vAxisTitle) {
        var dataTable = google.visualization.arrayToDataTable(data);

        var options = {
            title: title,
            curveType: 'function',
            legend: { position: 'bottom' },
            hAxis: {
                title: 'Date',
                format: 'MMM d, yyyy HH:mm:ss',
                gridlines: { count: 15 } // Adjust to show more gridlines
            },
            vAxis: {
                title: vAxisTitle
            }
        };

        var chart = new
google.visualization.LineChart(document.getElementById(elementId));
        chart.draw(dataTable, options);
    }
```

```
        refreshLineChartData();
        setInterval(refreshLineChartData, 10000); // Refresh data every 10 seconds
    }

    $(window).resize(function(){
        drawTemperatureChart();
        drawHumidityChart();
        drawLineCharts();
    });
</script>
</body>
</html>
```

nodemcu-dht22-mysql.ino

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClient.h>
#include <DHT.h>
#include <MQ135.h>

#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>

#define DHTPIN D3
#define DHTTYPE DHT22
#define PIN_MQ135 A0

DHT dht(DHTPIN, DHTTYPE);
MQ135 gasSensor = MQ135(PIN_MQ135);

#define CHAT_ID "826634550"
#define BOTtoken "7466435237:AAFE0uHD5iuGDG_Q1MmuDT-sRhankfc95PA"
```

```
X509List cert(TELEGRAM_CERTIFICATE_ROOT);
WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);
//Checks for new messages every 1 second.
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;



const char* ssid     = "Rumah 1 Malaysia-Maxis Fibre";
const char* password = "Waz5S8mJ18";

const char* SERVER_NAME = "http://samaq.atwebpages.com/sensordata.php";
String PROJECT_API_KEY = "tempQuality";

unsigned long lastMillis = 0;
long interval = 30000; // interval in milliseconds (30 seconds)

String getReadings(){
  float temperature, humidity, gas;
  temperature = dht.readTemperature();
  humidity = dht.readHumidity();
  gas = gasSensor.getCorrectedPPM(temperature, humidity);
  String message = "Temperature: " + String(temperature) + " ºC ";
  if(temperature<22.29)
    message += "(Too Cold) \n";
  else if(temperature>=22.29 && temperature<26.51)
    message += "(Cold) \n";
  else if(temperature>=26.51 && temperature<30.79)
    message += "(Normal) \n";
  else
    message += "(Hot) \n";
  message += "Humidity: " + String (humidity) + " % ";
```

```
  if(humidity<72.0)
    message += "(Low) \n";
  else if(humidity>=72.0 && humidity<87.0)
    message += "(Normal) \n";
  else
    message += "(High) \n";
  message += "C02 Concentration: " + String (gas) + " ppm \n";
  return message;
}

float gasReadings[2] = {0.0, 0.0}; // Array to store the two latest gas readings

String alertGas() {
    float temperature, humidity;
    temperature = dht.readTemperature();
    humidity = dht.readHumidity();

    // Read latest gas sensor reading
    float latestGas = gasSensor.getCorrectedPPM(temperature, humidity);

    // Shift previous reading to index 1
    gasReadings[1] = gasReadings[0];

    // Store latest reading at index 0
    gasReadings[0] = latestGas;

    // Calculate the difference between the two gas readings
    float diffgas = gasReadings[1] - gasReadings[0];

    // Prepare the message based on gas difference
    String message = "";
    if (diffgas > 40) {
        message += "Alert! CO2 dropping!";
```

```
    }

    return message;
}



//Handle what happens when you receive new messages
void handleNewMessages(int numNewMessages) {
  Serial.println("handleNewMessages");
  Serial.println(String(numNewMessages));

  for (int i=0; i<numNewMessages; i++) {
    // Chat id of the requester
    String chat_id = String(bot.messages[i].chat_id);
    if (chat_id != CHAT_ID){
      bot.sendMessage(chat_id, "Unauthorized user", "");
      continue;
    }

    // Print the received message
    String text = bot.messages[i].text;
    Serial.println(text);

    String from_name = bot.messages[i].from_name;

    if (text == "/start") {
      String welcome = "Welcome, " + from_name + ".\n";
      welcome += "Use the following command to get current readings.\n\n";
      welcome += "/readings: To check on temp, humidity and C02 \n";
      bot.sendMessage(chat_id, welcome, "");
    }
```

```
    if (text == "/readings") {
      String readings = getReadings();
      bot.sendMessage(chat_id, readings, "");
    }
  }
}

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

    Serial.println("Connecting to WiFi");
    configTime(0, 0, "pool.ntp.org");      // get UTC time via NTP
    client.setTrustAnchors(&cert); // Add root certificate for api.telegram.org
    dht.begin();

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

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

void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        if (millis() - lastMillis > interval) {
            float temperature = dht.readTemperature();
            float humidity = dht.readHumidity();
```

```
        if (isnan(temperature) || isnan(humidity)) {
          Serial.println("Failed to read from DHT sensor!");
        } else {

          float gas_level = gasSensor.getCorrectedPPM(temperature, humidity);
          // Assuming MQ135 analog output is connected to A0
          //float gas_level = analogRead(A0);
          int air_quality = getAirQuality(gas_level);

          sendSensorData(temperature, humidity, gas_level, air_quality);

          // Check gas levels and send alert if necessary
          String gasAlert = alertGas();
          if (!gasAlert.isEmpty())
            bot.sendMessage(CHAT_ID, gasAlert, "");
        }

        lastMillis = millis();
      }
    } else {
      Serial.println("WiFi Disconnected");
      WiFi.begin(ssid, password); // Reconnect to WiFi if disconnected
    }

    delay(2000); // Delay between sensor readings
  if (millis() > lastTimeBotRan + botRequestDelay)  {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

    while(numNewMessages) {
      Serial.println("got response");
      handleNewMessages(numNewMessages);
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
```

```
    }
    lastTimeBotRan = millis();
  }
}

int getAirQuality(float gas_level) {
    // Implement your logic to determine air quality based on gas level
    // Example logic:
    if (gas_level <=1000) {
        return 1; // High
    } else if (gas_level >1000 && gas_level<=2000) {
        return 2; // Moderate
    } else {
        return 3; // Low
    }
}

void sendSensorData(float temperature, float humidity, float gas_level, int air_quality) {
    WiFiClient client;
    HTTPClient http;

    String postData = "api_key=" + PROJECT_API_KEY;
    postData += "&temperature=" + String(temperature, 2);
    postData += "&humidity=" + String(humidity, 2);
    postData += "&gas_level=" + String(gas_level, 2);
    postData += "&air_quality=" + String(air_quality);

    http.begin(client, SERVER_NAME);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");

    int httpResponseCode = http.POST(postData);

    if (httpResponseCode > 0) {
```

```
    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);
  } else {
    Serial.print("Error in HTTP request: ");
    Serial.println(httpResponseCode);
  }

  http.end();
}
```

Links

Github: https://github.com/msolehab/DHT22_MQ135_ESP8266_Trend_Analysis/tree/main