

PROJECT 1: AIR QUALITY MONITORING

PHASE 1: PROBLEM DEFINITION:

The project involves developing an IoT-based air quality monitoring system with the primary objective of providing real-time air quality data to the public through an accessible online platform. The goal is to create awareness about the impact of air quality on public health and the environment.

KEY OBJECTIVES:

1. **Data Collection:** Deploy IoT devices equipped with air quality sensors to measure critical parameters, including CO, NO₂, SO₂, O₃, and VOCs.
2. **Real-Time Monitoring:** Install these devices strategically across the target area to continuously monitor air quality.
3. **Data Transmission:** Establish efficient data transmission protocols for sending sensor data to a central server.
4. **User-Friendly Platform:** Develop an intuitive web or mobile platform where the public can access real-time and historical air quality data.
5. **Data Visualization:** Implement data visualization tools to present air quality information clearly.
6. **Alerting System:** Create an alerting mechanism to notify users when air quality levels exceed predefined thresholds.
7. **Public Engagement:** Encourage public engagement by providing educational content on the platform, explaining the significance of air quality, and promoting community involvement.
8. **Scalability:** Design the system to accommodate future expansion with additional sensors and monitoring locations.
9. **Python Integration:** Utilize Python for data analysis, processing, and platform development.

DESIGN THINKING:

1. **Understand User Needs:** Begin by understanding the perspectives and needs of potential users, including the public, environmentalists, and authorities. Explore the air quality challenges in the target area.
2. **Define the Problem:** Clearly define the core problem and project objectives based on user insights and identified pain points related to accessing air quality information.
3. **Generate Ideas:** Brainstorm creative solutions for the IoT monitoring system and the awareness platform, involving diverse team members in the ideation process.

4. **Prototype:** Build a simple prototype of the IoT device and user interface, focusing on core functionalities to gather feedback.
5. **Test and Iterate:** Collect user feedback during usability tests with the prototype and make necessary improvements based on insights.
6. **Develop the Solution:** Create the complete IoT-based monitoring system and user platform, aligning with defined objectives and user needs.
7. **Deploy and Launch:** Implement the system, deploying IoT devices and launching the user platform to make real-time air quality data accessible to the public.
8. **Evaluate and Improve:** Continuously monitor system performance, gather user feedback, and make ongoing improvements based on data analytics and stakeholder input.
9. **Iterate and Adapt:** Keep the user at the centre of decision-making, be open to change, and consider expansion or additional features as needed to effectively raise public awareness about air quality.

IMPORTANT COMPONENTS:

Here are the important components for an IoT-based air quality monitoring system:

1. Sensor Devices
2. Data Transmission and Communication
3. Central Data Management System
4. Data Analysis and Processing
5. Alerting System
6. User Interface and Visualization
7. Geospatial Mapping

Phase 2: INNOVATION

After thorough research and analysis, we arrived at an innovative solution to solve the above problem as detailed in phase 1 of our project.

- We will be using the ESP32 micro controller as well as Arduino UNO microcontroller as both these suit the best for our project.
- We made this choice because we only require data on the concentrations of CO₂, NO₂ and smoke in the desired atmosphere to be posted on a public platform.

- **SENSOR**
 - We use MQ135 sensor in our air quality monitoring system because it can effectively detect CO2, NO2 gases and smoke and provides valuable data for comprehensive air quality assessment ensuring environmental safety.
 - Temperature and humidity sensor is used.
- **CONNECTIVITY**

Wi-Fi as it enables real time data transmission, allowing us to monitor air quality remotely.
- **CLOUD**

We use Bleeceptor it ensures scalability, data storage and analytics.
- **PROTOCOL**

HTTP: These are widely used protocols for transmitting data over the internet which are easy to implement and are supported by almost all web servers.
- **PUBLIC PLATFORM**

We are going to design a website for Air quality monitoring system.

PHASE 3

PROBLEM:

To develop the python script on IoT devices as per the project requirement.

SOLUTION:

An Arduino-based air quality monitoring system offers real-time data on pollutants like CO2 and NO2, temperature, humidity, and particulate matter. This information is transmitted to a central server and displayed through a user-friendly platform, enabling informed decisions, alerts, and fostering environmental awareness and community involvement.

SOURCE CODE:

```
#define BLYNK_TEMPLATE_ID "TMPLwToQUqRw"

#define BLYNK_TEMPLATE_NAME "Air Quality Monitoring"

#define BLYNK_AUTH_TOKEN "7kuX0IEEPHLVRSK2Jhgf81qpCgL3D0Nr"

#define BLYNK_PRINT Serial

#include <WiFi.h>

#include <BlynkSimpleEsp32.h>
```

```

#include <DHT.h>

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16,
2); byte degree_symbol[8] =

    {

        0b00111,

        0b00101,

        0b00111,

        0b00000,

        0b00000,

        0b00000,

        0b00000,

        0b00000

    };

Char auth[] = BLYNK_AUTH_TOKEN;

Char ssid[] = "Wokwi-GUEST"; // type your wifi name

Char pass[] = ""; // type your wifi password

BlynkTimer timer;

Int gas = 32;

Int sensorThreshold = 100;

#define DHTPIN 2 //Connect Out pin to D2 in NODE MCU

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

Void sendSensor()

{

    Float h = dht.readHumidity();

    Float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit

```

```
    If (isnan(h) || isnan(t)) {

    Serial.println("Failed to read from DHT sensor!");

    Return;

}

    Int analogSensor = analogRead(gas);

    Blynk.virtualWrite(V2, analogSensor);

    Serial.print("Gas Value: ");

    Serial.println(analogSensor);

    // You can send any value at any time.

    // Please don't send more that 10 values per second.

    Blynk.virtualWrite(V0, t);

    Blynk.virtualWrite(V1, h);

    Serial.print("Temperature : ");

    Serial.print(t);

    Serial.print("  Humidity : ");

    Serial.println(h);

}

Void setup()

{

    Serial.begin(115200);

    //pinMode(gas, INPUT);

    Blynk.begin(auth, ssid, pass);

    Dht.begin();

    Timer.setInterval(30000L, sendSensor);

    //Wire.begin();

    Lcd.begin(16,2);

    // lcd.backlight();
```

```
// lcd.clear();

Lcd.setCursor(3,0);

Lcd.print("Air Quality");

Lcd.setCursor(3,1);

Lcd.print("Monitoring");

Delay(2000);

Lcd.clear();

}

Void loop()

{

  Blynk.run();

  Timer.run();

  Float h = dht.readHumidity();

  Float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit

  Int gasValue = analogRead(gas);

  Lcd.setCursor(0,0);

  Lcd.print("Temperature ");

  Lcd.setCursor(0,1);

  Lcd.print(t);

  Lcd.setCursor(6,1);

  Lcd.write(1);

  Lcd.createChar(1, degree_symbol);

  Lcd.setCursor(7,1);

  Lcd.print("C");

  Delay(4000);

  Lcd.clear();

  Lcd.setCursor(0, 0);
```

```
Lcd.print("Humidity ");  
  
Lcd.print(h);  
  
Lcd.print("%");  
  
Delay(4000);  
  
Lcd.clear();  
  
//lcd.setCursor(0,0);  
  
// lcd.print(gasValue);  
  
// lcd.clear();  
  
Serial.println("Gas Value");  
  
Serial.println(gasValue);  
  
If(gasValue<1200)  
{  
  
    Lcd.setCursor(0,0);  
  
    Lcd.print("Gas Value: ");  
  
    Lcd.print(gasValue);  
  
    Lcd.setCursor(0, 1);  
  
    Lcd.print("Fresh Air");  
  
    Serial.println("Fresh Air");  
  
    Delay(4000);  
  
    Lcd.clear();  
  
}  
  
Else if(gasValue>1200)  
{  
  
    Lcd.setCursor(0,0);  
  
    Lcd.print(gasValue);  
  
    Lcd.setCursor(0, 1);  
  
    Lcd.print("Bad Air");
```

```
Serial.println("Bad Air");

Delay(4000);

Lcd.clear();

}

If(gasValue > 1200){

  //Blynk.email(karthiaenit@gmail.com, "Alert", "Bad Air!");

  Blynk.logEvent("pollution_alert","Bad Air");

}

}
```

INCLUDE LIBRARIES:

Here are the libraries used in the code:

- **WiFi.h:** This library is used for connecting the ESP32 to a Wi-Fi network.
- **BlynkSimpleEsp32.h:** This library is used for integrating the ESP32 with the Blynk IoT platform.
- **DHT.h:** This library is used for interfacing with DHT (Digital Humidity and Temperature) sensors such as the DHT11.
- **LiquidCrystal_I2C.h:** This library is used for controlling and interfacing with the I2Cconnected Liquid Crystal Display (LCD).

INITIALIZE OBJECTIVES AND VARIABLES:

- Initialize Blynk template ID, name, and authentication token.
- Create a BlynkTimer object for periodic sensor updates.
- Define variables for gas sensor, sensor threshold, and DHT sensor pins.
- Initialize LCD with custom character.
- Set Wi-Fi credentials and Blynk authentication token.
- Initialize DHT sensor.
- Define variables for temperature, humidity, and gas readings.
- Set up LCD display and print welcome message.
- In the loop, run Blynk and timer, read DHT and gas sensor data, and display it on the LCD.
- Log events with Blynk for air quality alerts.

Setup() Function:

- Initializes the Arduino and serial communication for debugging.
- Attempts to connect to a Wi-Fi network with the provided SSID and password.
- Continues to print "Connecting to WiFi..." until a successful connection is established.
- Once connected to Wi-Fi, it prints "Connected to WiFi".

Loop() Function:

- Continuously runs in a loop to collect sensor data and send it to the server.
- Reads CO2, NO2, and smoke sensor values using the readCO2() method from the MQ135 sensor library.
- Calls the sendDataToServer() function with the sensor readings.

SendDataToServer() Function:

- Prepares data in the form of a string containing CO2, NO2, and smoke values.
- Uses the WokwiHTTPClient library (for simulation purposes) to make an HTTP POST request to the specified serverURL.
- Adds the "Content-Type" header to specify the data format.
- Checks the HTTP response code. If the code is greater than zero, it means a successful response.
- If successful, it prints the HTTP response to the serial monitor. Otherwise, it prints an HTTP error.
- Closes the HTTP connection with http.end().
- Overall, this code initializes the Wi-Fi connection, reads air quality sensor data, sends this data to a server via HTTP POST, and provide feedback via the serial monitor. It's designed for simulation purposes, but with the appropriate server setup, it could be adapted for realworld air quality monitoring.

SIMULATION LINK:

<https://wokwi.com/projects/378756614608888833>

SIMULATION IMAGES

10/16/23, 8:49 PM

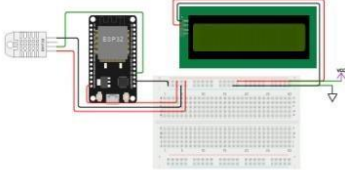
air quality monitoring - Wokwi ESP32, STM32, Arduino Simulator

air quality monitoring

sketch.ino diagram.json libraries.txt Library Manager

```
1  #define BLYNK_TEMPLATE_ID "TMPLwToQlqRw"
2  #define BLYNK_TEMPLATE_NAME "Air Quality Monitoring"
3  #define BLYNK_AUTH_TOKEN "7kuX0IEEPHLVRSK23hgF8lpCgI3D00lr"
4
5  #define BLYNK_PRINT Serial
6  #include <WiFi.h>
7  #include <BlynkSimpleEsp32.h>
8
9  #include <DHT.h>
10
11 #include <LiquidCrystal_I2C.h>
12
13 LiquidCrystal_I2C lcd(0x27, 16, 2);
14
15 byte degree_symbol[8] =
16 {
17     0b00111,
18     0b00101,
19     0b00111,
20     0b00000,
21     0b00000,
22     0b00000,
23     0b00000,
24     0b00000
25 };
26
27 char auth[] = BLYNK_AUTH_TOKEN;
28
29 char ssid[] = "Wokwi-GUEST"; // type your wifi name
30 char pass[] = ""; // type your wifi password
31
32 BlynkTimer timer;
33
34 int gas = 32;
```

Simulation



599
Fresh Air
Gas Value
824
Fresh Air
Gas Value: 781
Temperature : -1.00 Humidity : 15.40

PHASE 4: WEB DEVELOPMENT

In this technology project we will continue building our project by developing the platform as per project requirement.

The platform we are going to develop is through Bleeceptor cloud with HTTP protocol.

CODE :

```
#define BLYNK_TEMPLATE_ID "TMPLwToQUqRw"
#define BLYNK_TEMPLATE_NAME "Air Quality Monitoring"
#define BLYNK_AUTH_TOKEN "7kuX0IEEPHLVRSK2Jhgf81qpCgL3D0Nr"
#define BLYNK_PRINT Serial

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>
#include <LiquidCrystal_I2C.h>
#include <HTTPClient.h>
#include <WiFiClient.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

byte degree_symbol[8] = {
  0b00111,
  0b00101,
  0b00111,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
  0b00000,
```

```

0b000000
};
char auth[] = BLYNK_AUTH_TOKEN;
char ssid[] = "Wokwi-GUEST"; // type your WiFi name
char pass[] = ""; // type your WiFi password
BlynkTimer timer;

int gas = 32;
int sensorThreshold = 100;

#define DHTPIN 2 // Connect Out pin to D2 in NODE MCU
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// BEECEPTOR endpoint URL
const char* beeceptorURL = "https://akmns.free.beeceptor.com";

void sendDataToBeeceptor(float temperature, float humidity, int gasValue) {
  HTTPClient http;

  // Build the JSON payload
  String payload = "{\"temperature\":\"" + String(temperature, 2) +
    "\",\"humidity\":\"" + String(humidity, 2) +
    "\",\"gasValue\":\"" + String(gasValue) + "\"}";

  // Send the POST request to Beeceptor
  http.begin(beeceptorURL);
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST(payload);
  if (httpResponseCode > 0) {
    Serial.print("HTTP Response Code: ");
    Serial.println(httpResponseCode);
  }
}

```

```
String response = http.getString();
Serial.println(response);
} else {
    Serial.print("HTTP Error: ");
    Serial.println(httpResponseCode);
}
http.end();
}

void sendSensor() {
    float h = dht.readHumidity();

    float t = dht.readTemperature(); // or dht.readTemperature(true) for
    Fahrenheit
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }

    int analogSensor = analogRead(gas);
    Blynk.virtualWrite(V2, analogSensor);
    Serial.print("Gas Value: ");
    Serial.println(analogSensor);
    Blynk.virtualWrite(V0, t);
    Blynk.virtualWrite(V1, h);
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.print(" Humidity: ");
    Serial.println(h);
```

```
// Send data to BEECEPTOR
sendDataToBeeceptor(t, h, analogSensor);
}

void setup() {
  Serial.begin(115200);
  Blynk.begin(auth, ssid, pass);
  dht.begin();
  timer.setInterval(30000L, sendSensor);
  lcd.begin(16, 2);
  lcd.setCursor(3, 0);
  lcd.print("Air Quality");
  lcd.setCursor(3, 1);
  lcd.print("Monitoring");
  delay(2000);
  lcd.clear();
}

void loop() {
  Blynk.run();
  timer.run();

  float h = dht.readHumidity();
  float t = dht.readTemperature(); // or dht.readTemperature(true) for
  Fahrenheit
  int gasValue = analogRead(gas);
  lcd.setCursor(0, 0);
  lcd.print("Temperature ");
  lcd.setCursor(0, 1);
```

```
lcd.print(t);  
lcd.setCursor(6, 1);  
lcd.write(1);  
lcd.createChar(1, degree_symbol);  
lcd.setCursor(7, 1);  
lcd.print("C");  
delay(4000);  
lcd.clear();  
lcd.setCursor(0, 0);  
lcd.print("Humidity ");  
lcd.print(h);  
lcd.print("%");  
delay(4000);  
lcd.clear();  
Serial.println("Gas Value");  
Serial.println(gasValue);  
if (gasValue < 1200) {  
    lcd.setCursor(0, 0);  
    lcd.print("Gas Value: ");  
    lcd.print(gasValue);  
    lcd.setCursor(0, 1);  
    lcd.print("Fresh Air");  
    Serial.println("Fresh Air");  
    delay(4000);  
    lcd.clear();  
} else if (gasValue > 1200) {
```

```
lcd.setCursor(0, 0);  
lcd.print(gasValue);  
lcd.setCursor(0, 1);  
lcd.print("Bad Air");  
Serial.println("Bad Air");  
delay(4000);  
lcd.clear();  
}  
if (gasValue > 1200) {  
  // Blynk.email(karthiaenit@gmail.com, "Alert", "Bad Air!");  
  Blynk.logEvent("pollution_alert", "Bad Air");  
}  
}
```

WOKWI SIMULATION LINK WITH BEECEPTOR CLOUD:

<https://wokwi.com/projects/379988279376105473>

TEAM MEMBERS:

- | | |
|-----------------|------------|
| 1. A.Arthi | 2021504001 |
| 2. T.Karthiaeni | 2021504012 |
| 3. G.Mythili | 2021504024 |
| 4. R.Narmatha | 2021504026 |
| 5. R.Sneka | 2021504043 |

PHASE-5:

The components we used in our projects are:

- 1.ESP32
- 2.DHT11 sensor(Temperature and Humidity sensor)
- 3.LiquidCrystal_I2C
- 4.Bread Board

1.ESP 32:

The ESP32 is a microcontroller developed by Espressive Systems, succeeding the ESP8266, known for its enhanced capabilities and versatility. It features a dual-core Xtensa LX6 microcontroller with a 32-bit architecture, capable of running at speeds up to 240 MHz. In our project we use esp32 to built-in Wi-Fi capabilities allow for easy data transmission. This is crucial for sending real-time air quality information to a central server or for remote monitoring.

2.DHT11:

The DHT11 sensor is capable of measuring both temperature and humidity levels in the surrounding environment. It provides relatively accurate readings for both temperature and humidity, though it may not be as precise as some more specialized sensors. The DHT11 is a digital sensor, which means it provides a digital output signal (in the form of a digital signal HIGH or LOW) that can be easily read by a microcontroller or a computer.

3.LiquidCrystal_I2C:

The Liquid Crystal I2C module serves as a crucial component in an air quality monitoring system project. It interfaces with microcontrollers via the I2C communication protocol, allowing for easy display of information. The LCM1602 employs a 16x2 character Liquid Crystal Display (LCD), providing a clear and readable output for presenting air quality data. Its I2C interface simplifies the wiring and reduces the number of pins required for connection, streamlining integration with microcontrollers like the ESP32. By incorporating the LCM1602 module, the air quality monitoring system can offer real-time visual feedback to users, displaying critical air quality parameters such as temperature, humidity, and pollutant levels.

IMAGES TAKEN FROM WOKWI:

AIR QUALITY IS BAD:

WOKWI [SAVE](#) [SHARE](#) [air quality monitoring Copy](#) Docs [SIGN IN](#)

sketch.ino diagram.json libraries.txt Library Manager

```
124 delay(4000);
125 lcd.clear();
126 lcd.setCursor(0, 0);
127 lcd.print("Humidity ");
128 lcd.print(h);
129 lcd.print("%");
130 delay(4000);
131 lcd.clear();
132
133 Serial.println("Gas Value");
134 Serial.println(gasValue);
135
136 if (gasValue < 1200) {
137   lcd.setCursor(0, 0);
138   lcd.print("Gas Value: ");
139   lcd.print(gasValue);
140   lcd.setCursor(0, 1);
141   lcd.print("Fresh Air");
142   Serial.println("Fresh Air");
143   delay(4000);
144   lcd.clear();
145 } else if (gasValue > 1200) {
146   lcd.setCursor(0, 0);
147   lcd.print("Gas Value: ");
148   lcd.print(gasValue);
149   lcd.setCursor(0, 1);
150   lcd.print("Bad Air");
151   Serial.println("Bad Air");
152   delay(4000);
153   lcd.clear();
154 }
155
156 if (gasValue > 1200) {
157   // Blynk.email(karthiaenit@gmail.com, "Alert", "Bad Air!");
158   Blynk.logEvent("pollution_alert", "Bad Air");
159 }
160
```

Simulation

06:32.167 76%

1668
Bad Air
Gas Value: 1704
Temperature: -1.00 Humidity: 15.40
HTTP Response Code: 200

Hey ya! Great to see you here. Btw, nothing is configured for this request

AIR QUALITY IS GOOD:

WOKWI [SAVE](#) [SHARE](#) [air quality monitoring Copy](#) Docs [SIGN UP](#)

sketch.ino diagram.json libraries.txt Library Manager

```
85 Serial.print(" Humidity: ");
86 Serial.println(h);
87
88 // Send data to Bseeceptor
89 sendDataToBseeceptor(t, h, analogSensor);
90 }
91
92 void setup() {
93   Serial.begin(115200);
94   Blynk.begin(auth, ssid, pass);
95   dht.begin();
96   timer.setInterval(3000L, sendSensor);
97   lcd.begin(16, 2);
98
99   lcd.setCursor(3, 0);
100   lcd.print("Air Quality");
101   lcd.setCursor(3, 1);
102   lcd.print("Monitoring");
103   delay(2000);
104   lcd.clear();
105 }
106
107 void loop() {
108   Blynk.run();
109   timer.run();
110
111   float h = dht.readHumidity();
112   float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit
113   int gasValue = analogRead(gas);
114
115   lcd.setCursor(0, 0);
116   lcd.print("Temperature ");
117   lcd.setCursor(0, 1);
118   lcd.print(t);
119   lcd.setCursor(6, 1);
120   lcd.write(1);
121   lcd.createChar(1, degree_symbol);
122 }

```

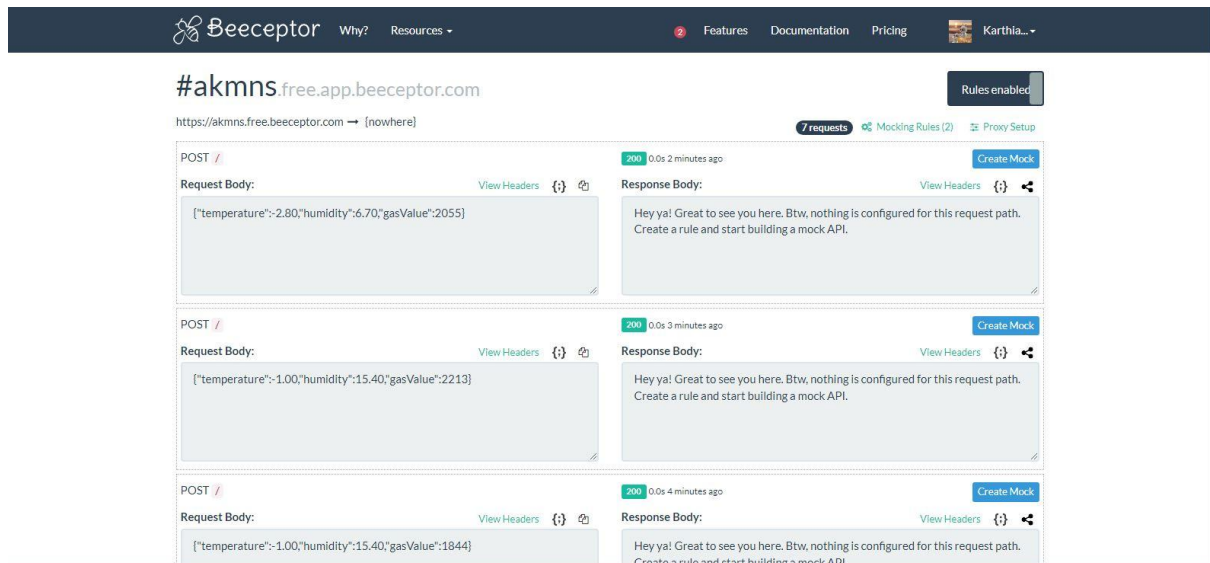
Simulation

05:05.227 99%

Fresh Air
Gas Value: 784
Temperature: -1.00 Humidity: 15.40
HTTP Response Code: 200

Hey ya! Great to see you here. Btw, nothing is configured for this request path. Create a rule and start building a mock API.

OUTPUT IMAGE FROM BEECEPTOR:



SIMULATION LINK FOR BEECEPTOR CLOUD :

<https://akmns.free.beeceptor.com/>

TEAM MEMBERS:

1. Aarthi 2021504001
2. T.Karthiaeni 2021504012
3. G.Mythili 2021504024
4. R.Narmatha 2021504026
5. R.Sneka 2021504043