IOT-BASED AIR QUALITY MONITORING SYSTEM

PHASE 4

DEVELOPMENT PART 2

It will be challenging to design an effective and high-performing system if we don't adhere to a procedure. We have developed a methodical procedure for developing a mobile IoT-based air quality monitoring system. This will also involve choosing the features and creating the design.

–

Figure 1

1. STUDY THE MARKET, COMPETITION AND TARGET USERS

- Analyze the air quality monitoring system market by assessing size, growth, trends, regulations, and competition.

- Competition in the air quality monitoring system industry centers around sensor technology advancements and the development of robust data analysis and integration capabilities.

- The general public and local communities, as well as government and regulatory agencies. The former seeks real-time air quality information for health and safety decisions, while the latter relies on this data for policy enforcement and public health protection.

2. DEFINE THE CORE PURPOSE OF THE PRODUCT

- The core purpose of an air quality monitoring system is to systematically assess air quality by measuring pollutants and particulate matter.

- It provides real-time data to the public for informed decisions on outdoor activities and safeguards public health.

- Government agencies use it to enforce standards and formulate environmental policies, contributing to overall environmental well-being.

3. VALIDATE THE CONCEPT AND DEFINE THE BLUEPRINT

- Air quality monitoring systems serve as indispensable tools for ensuring healthier and safer environments.

- The concept is validated by their ability to systematically collect, process, and analyze data on air quality, including the concentrations of pollutants and particulate matter.

- The blueprint of an air quality monitoring system encompasses a network of sensors strategically placed to measure various air quality parameters, including pollutants and environmental conditions.

- These sensors collect real-time data, which is transmitted to a central processing unit. The data undergoes calibration, validation, and noise filtering, ensuring accuracy.

4. DETERMINE THE UNIQUE FEATURES

- Air quality monitoring systems provide real-time, location-specific data on various pollutants, helping with pollution management and regulatory compliance. They offer visualized data, trigger alerts during pollution spikes, and store historical information for trend analysis.

- These systems also integrate with weather data, which aids in understanding environmental interactions. Portable units offer mobile monitoring, and citizen engagement is encouraged.

- The data generated supports research, policy development, and public awareness efforts, making these systems essential tools for safeguarding public health and the environment.

5. WORK WITH UI/UX TEAM AND CREATE A WORKABLE PROTOTYPE

- Collaborating with the UI/UX team, we'll create a user-friendly prototype for an air quality monitoring system.
- The design will focus on an intuitive interface with real-time data, clear Air Quality Index (AQI) visuals, and accessible web and mobile platforms.
- It will incorporate alert systems for safety, historical data analysis, and report generation.
- The design aims to deliver a clean, responsive layout to ensure a seamless and engaging experience for the public and regulatory agencies

6. DEVELOP THE DESIGNS USING CLEAN CODES

- Developing the design for the air quality monitoring system requires clean code practices.We'll employ a modular, well-documented approach with efficient algorithms for data processing, storage, and analysis.
- The code will be optimized for scalability and performance to handle large datasets. Security will be a priority, safeguarding sensitive air quality data.
- It will adhere to industry standards and best practices, promoting code reusability and maintainability. Additionally, the code will be thoroughly tested to identify and rectify any issues, ensuring a robust and reliable system. This approach will facilitate the system's long-term functionality and ease of future enhancements.

7. TEST THE APPLICATION BEFORE LAUNCHING

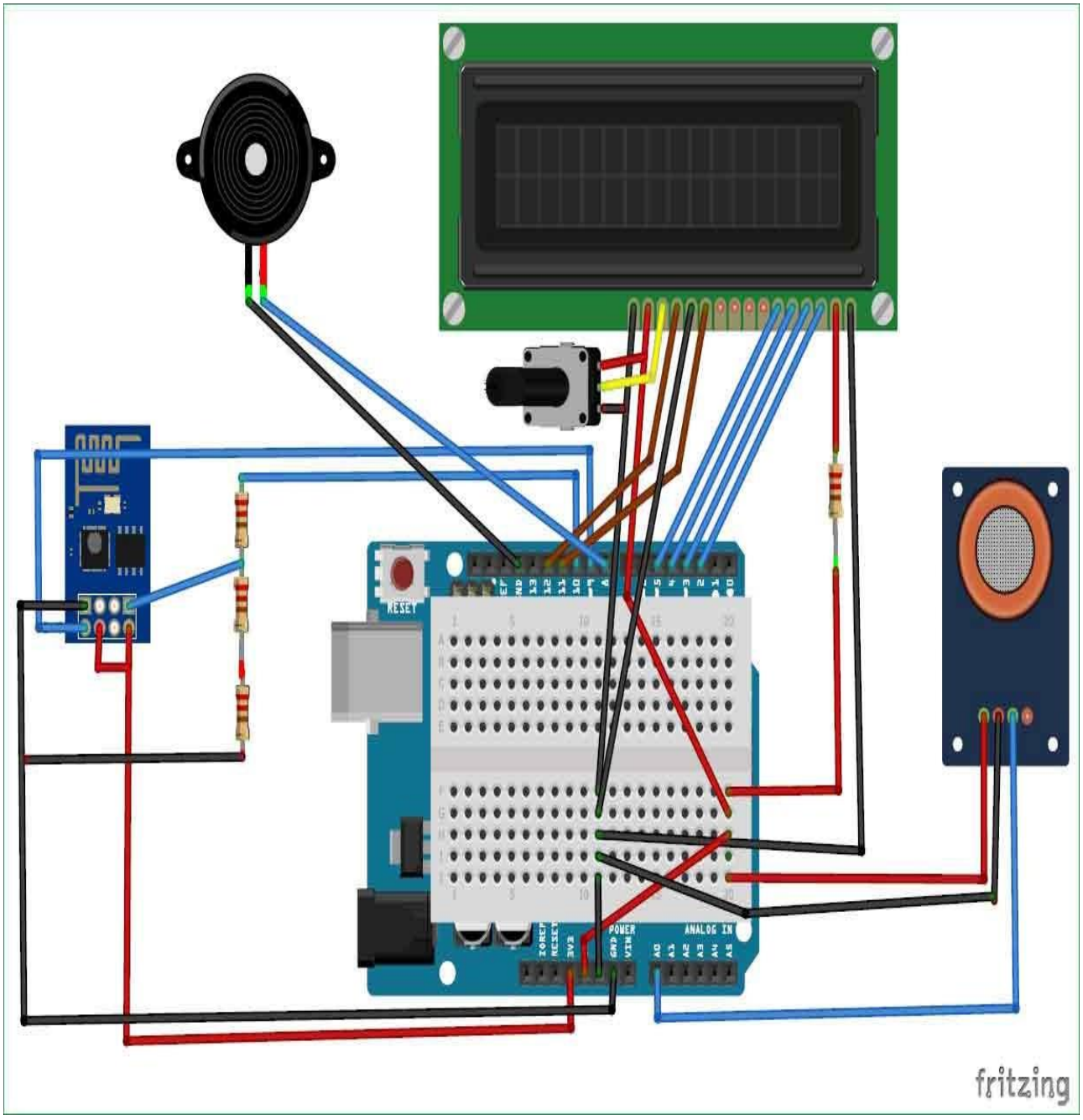
- Before launching the air quality monitoring system, comprehensive testing is essential. This includes unit testing to assess individual code components, integration testing for seamless system interactions, and system testing for overall functionality.
- Realistic scenarios and edge cases are considered to ensure system reliability and accuracy. Performance testing verifies the system's efficiency under anticipated loads.
- User acceptance testing gathers user feedback and ensures the system aligns with their needs. This multi-tiered testing process guarantees a robust, reliable system that meets user expectations, minimizing potential issues upon launch.

8. PLAN FOR AFTER-SALES SUPPORT

- The after-sales support plan for the air quality monitoring system comprises dedicated technical assistance, regular software updates, user documentation, and training resources.
- It includes maintenance and calibration services for hardware reliability. User feedback mechanisms are in place to gather input for system enhancement.
- This holistic approach ensures a seamless, reliable, and up-to-date experience for customers, promoting long-term satisfaction with the air quality monitoring system.

CONNECTION:

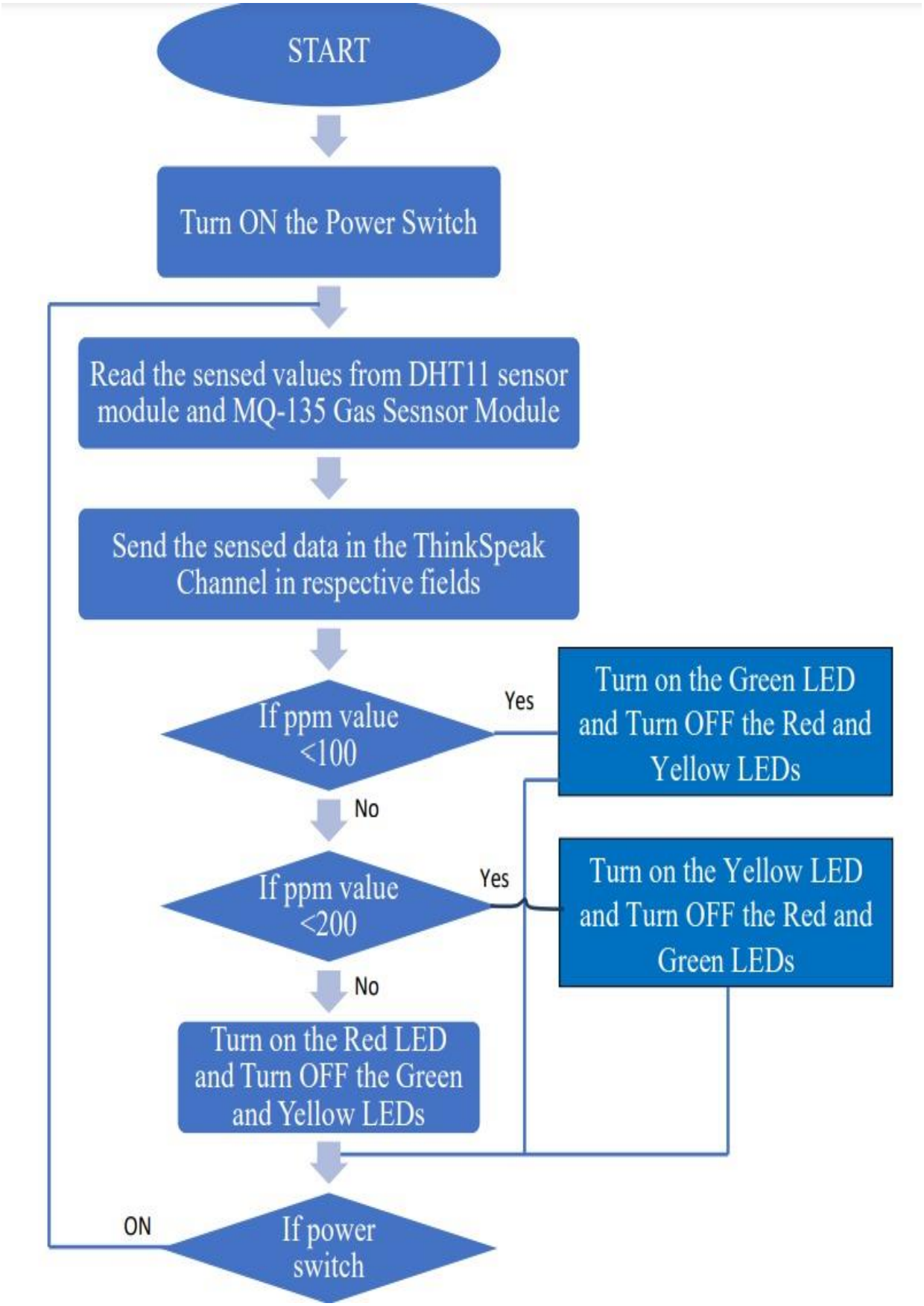
In IoT-based air quality monitoring, the key is the seamless integration of sensors, IoT devices, connectivity, and a central platform that allows users to access and make informed decisions based on real-time and historical air quality data. The effectiveness of such a system depends on the reliability and accuracy of the sensors and the robustness of the communication and data processing infrastructure.



PIN CONFIGURATION:

Pin Number	Pin Name	Pin Function
1	RESET	Active Low External Reset Signal
2	ADC(TOUT)	ADC Pin Analog Input
3	CH_PD	Active High Chip Enable
4	GPIO16	General purpose IO
5	GPIO14	General purpose IO
6	GPIO12	General purpose IO
7	GPIO13	General purpose IO
8	VCC	Power Supply
9	Ground	Ground
10	GPIO15	General purpose IO, should be connected to ground for booting from internal flash
11	GPIO1	General purpose IO, Serial Tx1
12	GPIO0	General purpose IO, Launch Serial Programming Mode if Low while Reset or Power ON
13	GPIO4	General purpose IO
14	GPIO5	General purpose IO
15	GPIO3	General purpose IO, Serial Rx
16	GPIO1	General purpose IO, Serial Tx

FLOW CHART:


START
Turn ON the Power Switch
Read the sensed values from DHT11 sensor module and MQ-135 Gas Sesnsor Module
Send the sensed data in the ThinkSpeak Channel in respective fields
If ppm value <100
Yes
Turn on the Green LED and Turn OFF the Red and Yellow LEDs
No
If ppm value <200
Yes
Turn on the Yellow LED and Turn OFF the Red and Green LEDs
No
Turn on the Red LED and Turn OFF the Green and Yellow LEDs
If power switch
ON

THING SPEAK CLOUD:

ThingSpeak is open-source software written in Ruby which allows users to communicate with internet-enabled devices. It facilitates data access, retrieval and logging of data by providing an API to both the devices and social network websites. ThingSpeak was originally launched by ioBridge in 2010 as a service in support of IoT applications. ThingSpeak has integrated support from the numerical computing software MATLAB from MathWorks, allowing ThingSpeak users to analyse and visualize uploaded data using MATLAB without requiring the purchase of a MATLAB license from MathWorks.

WORKING PROCEDURES:

NodeMCU plays the main controlling role in this project. It has been programmed in a manner,
such that, it senses the sensory signals from the sensors and shows the quality level via led indicators. The DHT11 sensor module is used to measure the temperature and the humidity of the
surroundings.With the help of the MQ-135 gas sensor module, air quality is measured in ppm. These data are fed to the ThinkSpeak cloud over the internet. We have also provided LED indicators to indicate the safety levels.

STEP 1 : Firstly, the calibration ofthe MQ-135 gas sensor module is done. The sensor is set to preheat for 24 minutes. Then the software code is uploaded to the NodeMCU followed by the hardware circuit to calibrate the sensor has been performed.
STEP 2 : Then, the DHT11 sensor is set to preheat for 10 minutes.
STEP 3 : The result of calibration found in STEP 1 is used to configure the final working code.
STEP 4 : The final working code is then uploaded to the NodeMCU.
STEP 5 : Finally, the complete hardware circuit is implemented.

The software codes and the hardware circuits are described in the following chapters

SOFTWARE CODE:

```
void setup()
{
Serial.begin(9600); //Baud rate
19 | P a g e
pinMode(A0,INPUT);
}
void loop()
{
 float sensor_volt; //Define variable for sensor voltage
 float RS_air; //Define variable for sensor resistance
```

```
 float R0; //Define variable for R0
 float sensorValue=0.0; //Define variable for analog readings
Serial.print("Sensor Reading = ");
Serial.println(analogRead(A0));
for(int x = 0 ; x < 500 ; x++) //Start for loop
 {
sensorValue = sensorValue + analogRead(A0); //Add analog values of sensor 500 times
 }
sensorValue = sensorValue/500.0; //Take average of readings
sensor_volt = sensorValue*(5.0/1023.0); //Convert average to voltage
RS_air = ((5.0*1.0)/sensor_volt)-1.0; //Calculate RS in fresh air
 R0 = RS_air/3.7; //Calculate R0
Serial.print("R0 = "); //Display "R0"
Serial.println(R0); //Display value of R0
delay(1000); //Wait 1 second
}
```

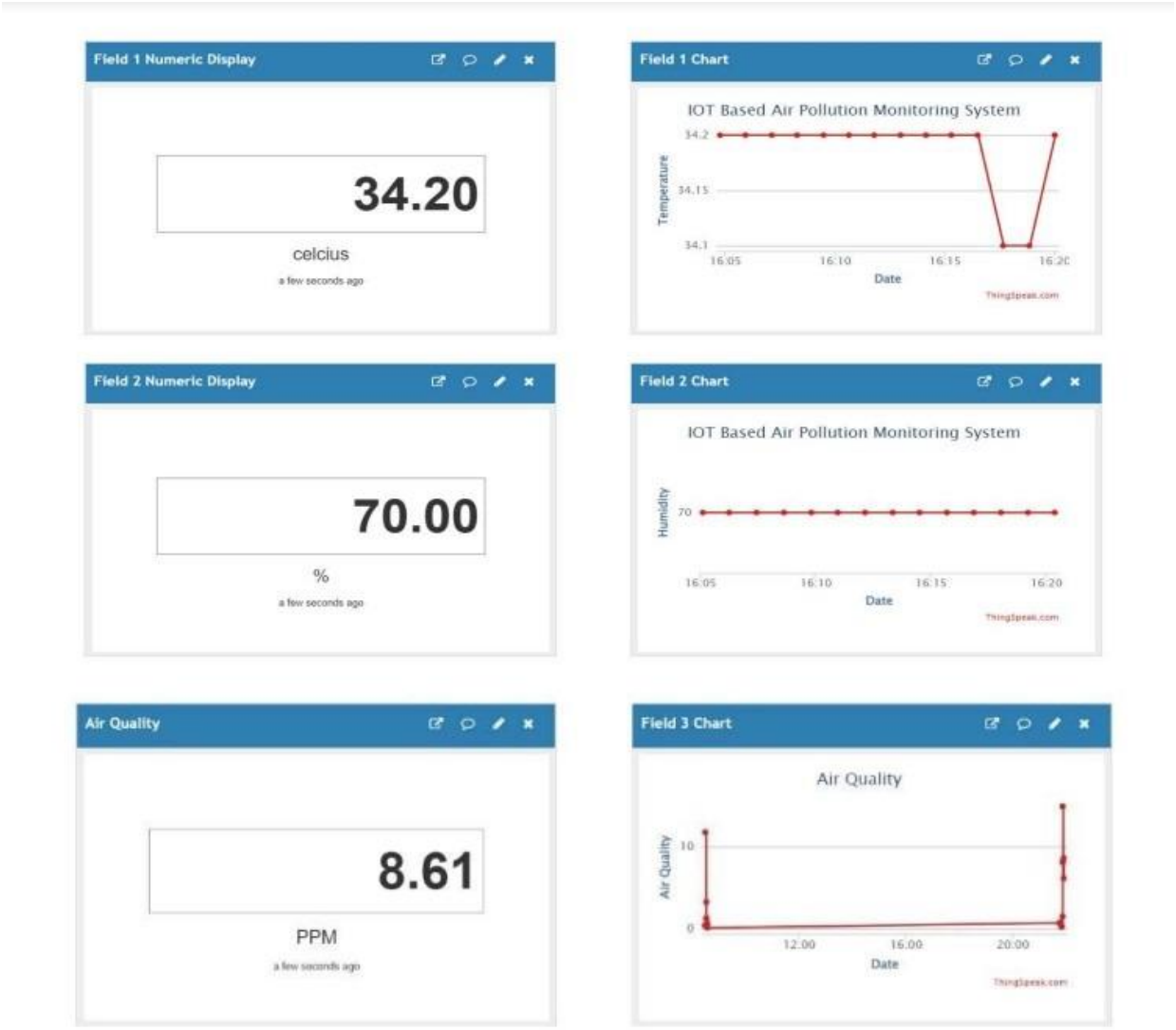
EXECUTION OF THE MAIN PROGRAM:

```
#include <ESP8266WiFi.h>
#include <DHT.h>
#include <ThingSpeak.h>
DHT dht(D5, DHT11);
#define LED_GREEN D2
#define LED_YELLOW D3
#define LED_RED D4
#define MQ_135 A0
int ppm=0;
float m = -0.3376; //Slope
float b = 0.7165; //Y-Intercept
float R0 = 3.12; //Sensor Resistance in fresh air from previous code
WiFiClient client;
long myChannelNumber = 123456; // Channel
const char myWriteAPIKey[] = "API_Key";
void setup() {
 // put your setup code here, to run once:
Serial.begin(9600);
pinMode(LED_GREEN,OUTPUT);
pinMode(LED_YELLOW,OUTPUT);
pinMode(LED_RED,OUTPUT);
pinMode(MQ_135, INPUT);
WiFi.begin("WiFi_Name", "WiFi_Password");
while(WiFi.status() != WL_CONNECTED)
```

```
 {
delay(200);
Serial.print(".");
 }
Serial.println();
Serial.println("NodeMCU is connected!");
Serial.println(WiFi.localIP());
dht.begin();
ThingSpeak.begin(client);
}
void loop() {
 float sensor_volt; //Define variable for sensor voltage
 float RS_gas; //Define variable for sensor resistance
 float ratio; //Define variable for ratio
 int sensorValue;//Variable to store the analog values from MQ-135
 float h;
 float t;
 float ppm_log; //Get ppm value in linear scale according to the the ratio value
 float ppm; //Convert ppm value to log scale
 h = dht.readHumidity();
delay(4000);
 t = dht.readTemperature();
delay(4000);
sensorValue = analogRead(gas_sensor); //Read analog values of sensor
sensor_volt = sensorValue*(5.0/1023.0); //Convert analog values to voltage
RS_gas = ((5.0*1.0)/sensor_volt)-1.0; //Get value of RS in a gas
 ratio = RS_gas/R0; // Get ratio RS_gas/RS_air
ppm_log = (log10(ratio)-b)/m; //Get ppm value in linear scale according to the ratio value
 ppm = pow(10, ppm_log); //Convert ppm value to log scale
Serial.println("Temperature: " + (String) t);
Serial.println("Humidity: " + (String) h);
Serial.println("Our desired PPM = "+ (String) ppm);
ThingSpeak.writeField(myChannelNumber, 1, t, myWriteAPIKey);
delay(20000);
ThingSpeak.writeField(myChannelNumber, 2, h, myWriteAPIKey);
delay(20000);
ThingSpeak.writeField(myChannelNumber, 3, ppm, myWriteAPIKey);
delay(20000);
 if(ppm<=100)
 {
digitalWrite(LED_GREEN,HIGH);
digitalWrite(LED_YELLOW,LOW);
digitalWrite(LED_RED,LOW);
 }
```

```
 else if(ppm<=200)
 {
digitalWrite(LED_GREEN,LOW);
digitalWrite(LED_YELLOW,HIGH);
digitalWrite(LED_RED,LOW);
 }
 else
 {
digitalWrite(LED_GREEN,LOW);
digitalWrite(LED_YELLOW,LOW);
digitalWrite(LED_RED,HIGH);
 }
delay(2000);}
```

OBSERVATION IN THINGSPEAK CLOUD:



THE PROBLEM AIR QUALITY MONITORING SYSTEM APP SOLVES:

THE OBJECTIVES OF THIS APP ARE AS FOLLOWS:

- Pollution Assessment: Continuously assess and quantify air pollutant levels.
- Public Health Protection: Provide real-time data and alerts to safeguard public health.
- Environmental Compliance: Ensure adherence to environmental regulations and standards.
- Data for Policy: Support the development of air quality policies and regulations.
- Trend Analysis: Track historical data to identify patterns and trends.
- Research Support: Provide data for scientific research on air pollution effects.
- Emergency Response: Assist in responding to air quality emergencies.
- Education and Public Awareness: Engage the public and raise awareness.
- Weather Interaction: Study weather's impact on air quality.
- Localized Interventions: Guide targeted pollution mitigation strategies.

THE FUNCTIONS OF THIS APP ARE AS FOLLOWS:

- An air quality monitoring system app is a valuable tool with numerous functions that help users stay informed about air quality, make health-conscious decisions, and plan activities based on current and forecasted conditions.
- First and foremost, these apps display real-time air quality data, presenting parameters such as PM2.5, PM10, O3, NO2, SO2, CO, and more. This data is typically presented through user-friendly visualizations like graphs and color-coded maps.
- Users can also access historical air quality data through the app, allowing them to track trends and compare current conditions to past records. The app often calculates the Air Quality Index (AQI), providing a standardized and easily understandable measurement of air quality. This helps users quickly assess the current situation and its impact on their health.
- One of the most valuable features is the ability to obtain location-specific data. Users can receive air quality information for their exact location based on GPS or manual input. This is particularly useful when planning outdoor activities or assessing local air quality in real time.
- The app may also offer health recommendations based on the current air quality, suggesting actions like staying indoors or wearing masks when air quality is poor. Users can set up push notifications to receive alerts when air quality significantly deteriorates, enabling them to take preventive actions promptly.
- Furthermore, these apps often include forecasts, allowing users to plan their activities based on expected air quality conditions. They can also show trends, indicating whether air quality is improving or worsening over time.
- Some apps offer integration with personal air quality sensors, educational content to enhance users' understanding, sharing and reporting features, customizable settings, and map overlays that help visualize air quality data in the context of a map.

Challenges we ran into:

- Data Accuracy: Advancements in sensor technology and calibration improve data accuracy.
- Remote Monitoring: Enables remote equipment maintenance and reduces physical visits.
- Data Integration: Improved mechanisms for comprehensive data integration and sharing.
- Data Interpretation: Data visualization tools and public awareness enhance understanding.
- Funding Sources: Diverse funding options support air quality monitoring.
- Privacy Regulations: Implementation of privacy regulations and data anonymization.
- Extreme Weather Preparedness: Robust equipment designs and protective measures.
- Data Security Measures: Cybersecurity protocols safeguard air quality data.
- Public Engagement: Campaigns promote awareness and citizen participation.
- Urban Planning Integration: Air quality data increasingly influences urban planning.
- Interference Mitigation: Improved sensor technology minimizes data interference.