Smart City Air Quality Network

Project Overview:

Provide a brief overview of the project, including its goals, objectives, and the importance of smart city air quality monitoring system through IoT.

Project Scope:

Define the boundaries of the project, including what will be included and what will not be included.

Project Team:

List the members of the project team, including their roles and responsibilities.

Technology Stack:

Outline the web development technologies and IoT technologies you plan to use in the project. This may include languages, frameworks, hardware components, and software tools.

System Architecture:

Provide a high-level overview of the system's architecture. This should include components like sensors, microcontrollers, data storage, and the web platform.

Environmental Parameters to Monitor:

List the environmental parameters that your IoT system will monitor. For example humidity,co2,gases, etc.

Hardware and Software Components:

Detail the specific hardware and software components you will use for data collection, processing, and storage.

Data Flow:

Explain the flow of data from the sensors to the cloud platform and how data will be processed and stored.

Web Platform Development:

Describe the web platform you will develop. Include details about its functionalities, features, and the technologies you'll use for web development.

User Interface (UI):

Provide information on the design and layout of the user interface, including any mockups or wireframes if available.

Data Visualization:

Explain how data will be visualized on the platform, including charts, graphs, and real-time updates.

Data Storage:

Describe how and where data will be stored, whether in a database, cloud storage, or other means.

Data Security:

Explain the security measures in place to protect data, including encryption and access control.

Alerts and Notifications:

Detail how users will receive alerts or notifications based on the data, such as email alerts or mobile app notifications.

User Roles:

Define different user roles and their access privileges within the web platform.

Testing and Quality Assurance:

Explain the testing strategies and quality assurance measures that will be implemented during the project.

Deployment Plan:

Describe the plan for deploying the IoT devices, sensors, and web platform in the field.

Maintenance and Support:

Outline the plan for ongoing maintenance, updates, and support after the project is deployed.

Timeline:

Provide a project timeline that includes milestones and deadlines.

Budget:

Estimate the budget required for the project, including hardware costs, software licenses, and personnel expenses.

Risks and Mitigations:

Identify potential risks and the strategies you'll use to mitigate them.

Conclusion:

Summarize the key points of the document and highlight the project's significance.

Appendices:

Include any additional documents or references that are relevant to the project, such as technical specifications, code snippets, or diagrams.

Set Up a Cloud Platform:

It's often a good practice to have a cloud platform as an intermediary yours IoT devices . Cloud platforms like Bylnk platform, or Google Cloud IoT Core provide the infrastructure to manage and process data from IoT devices.

Device Registration and Authentication:

Register your IoT devices on the cloud platform and set up authentication and security mechanisms. This typically involves generating API keys, certificates, or tokens to ensure secure communication.

Data Ingestion:

Configure the IoT devices to send data to the cloud platform using the chosen communication protocol. This data can include environmental parameters such as gases, humidity, or co2.

Real-time Data Display:

Design the mobile app's user interface to display real-time data from your IoT devices. This could include charts, graphs, or numerical values that update as new data arrives.

User Alerts and Notifications:

Implement push notifications or in-app alerts to inform users of significant environmental changes or system events. These notifications can be triggered based on the data received from the IoT devices.

User Registration and Management:

Allow users to register and manage their accounts within the Google accounts, including setting preferences and configuring alert thresholds.

Testing and Debugging:

Thoroughly test the mobile app to ensure that it can successfully communicate with the IoT devices and cloud platform. Debug any issues that arise during the testing phase.

Deployment:

Publish the mobile app on app stores (e.g., Apple App Store and Google Play Store) for users to download and install.

User Training and Documentation:

Provide users with training and documentation on how to use the mobile app and understand the environmental data it presents.

Maintenance and Updates:

Regularly maintain and update both the mobile app and the IoT devices' firmware to address bugs, add new features, and improve security.

Monitoring and Analytics:

Implement analytics and monitoring tools to track app usage and IoT device performance. This can help you identify issues and optimize the system.

Program:

Creating a C program for an smart city air quality network IoT project that connects to a mobile app involves several steps. I'll provide an example C programming code snippet for a simplified scenario to get you started. This example assumes you have a cloud-based IoT platform for data storage and retrieval. You can adapt this code to your specific project requirements.

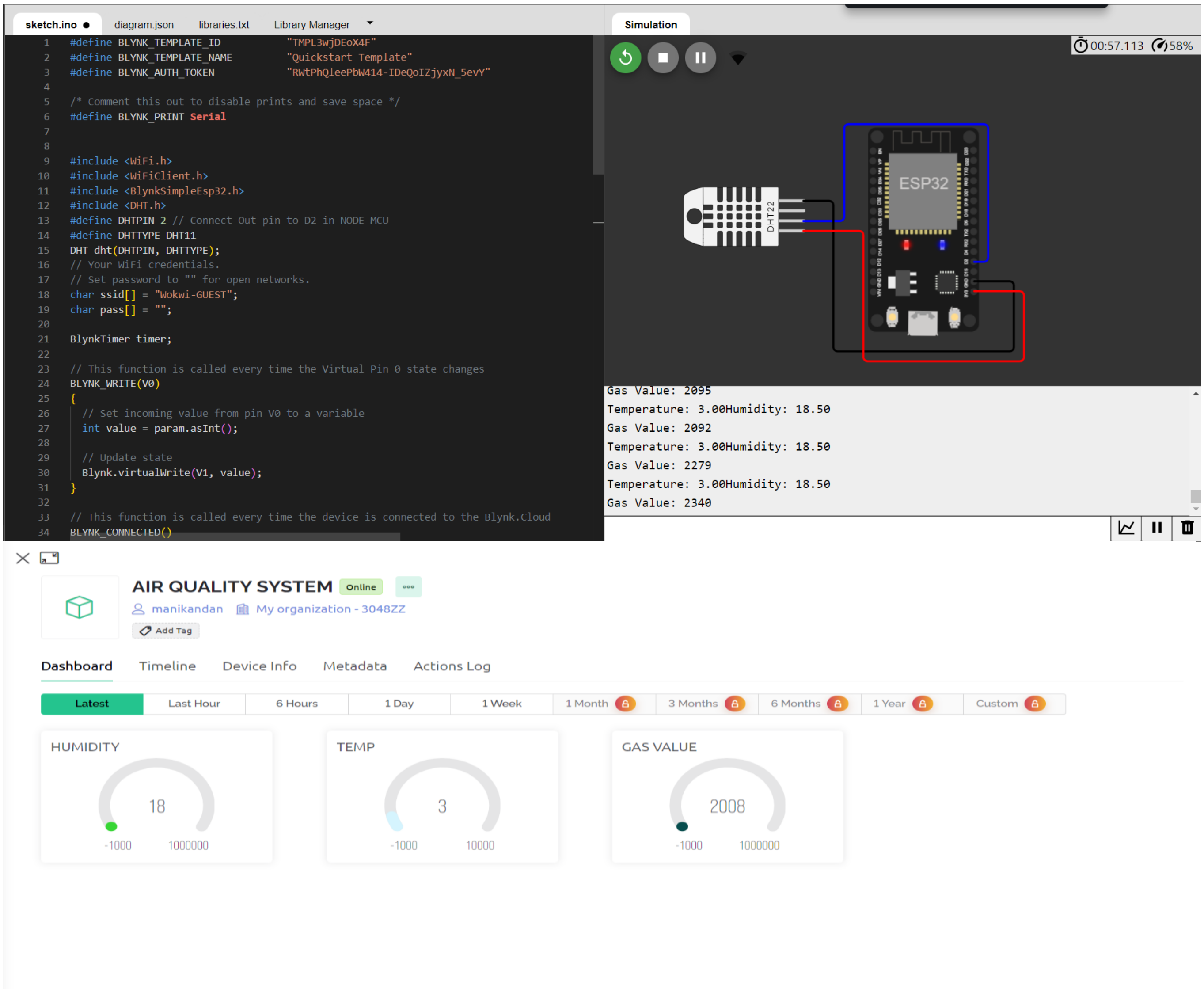
Code:

```
#define BLYNK_TEMPLATE_ID "TMPL3wjDEoX4F"

#define BLYNK_TEMPLATE_NAME "Quickstart Template"

#define BLYNK_AUTH_TOKEN "RWtPhQleePbW414-IDeQoIZjyxN_5evY"

/* Comment this out to disable prints and save space */

#define BLYNK_PRINT Serial

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <DHT.h>

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

#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

// Your WiFi credentials.

// Set password to "" for open networks.

char ssid[] = "Wokwi-GUEST";

char pass[] = "";

BlynkTimer timer;

// This function is called every time the Virtual Pin 0 state changes

BLYNK_WRITE(V0)

{

 // Set incoming value from pin V0 to a variable

 int value = param.asInt();

 // Update state

 Blynk.virtualWrite(V1, value);

}

// This function is called every time the device is connected to the Blynk.Cloud
```

```
BLYNK_CONNECTED()

{

 // Change Web Link Button message to "Congratulations!"

 Blynk.setProperty(V3, "offImageUrl",
"https://staticimage.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations.png");

 Blynk.setProperty(V3, "onImageUrl",
"https://staticimage.nyc3.cdn.digitaloceanspaces.com/general/fte/congratulations_pressed.png");

 Blynk.setProperty(V3, "url",
"https://docs.blynk.io/en/getting-started/what-do-ineed-to-blynk/how-quickstart-device-was-made"
);

}

// This function sends Arduino's uptime every second to Virtual Pin 2.

void myTimerEvent()

{

 // You can send any value at any time.

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

 Blynk.virtualWrite(V2, millis() / 1000);

}

void setup()

{

 // Debug console

 Serial.begin(115200);

 Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

 // You can also specify server:

 //Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass, "blynk.cloud", 80);

 //Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass, IPAddress(192,168,1,100),

8080);

 // Setup a function to be called every second

 timer.setInterval(1000L, myTimerEvent);

}

int gas = 32;

int sensorThreshold = 100;
```

```
void loop()
{
 Blynk.run();
 timer.run();
 // You can inject your own code or combine it with other sketches.
 // Check other examples on how to communicate with Blynk. Remember
 // to avoid delay() function!
 float h = dht.readHumidity();
 float t = dht.readTemperature();
 if (isnan(h) || isnan(t)) {
 Serial.println("Failed to read from DHT sensor!");
 return;
 }
 int gasValue = analogRead(gas);
 Blynk.virtualWrite(V2, gasValue); // Send gas value to Blynk
 Serial.print("Gas Value: ");
 Serial.println(gasValue);
 Blynk.virtualWrite(V0, t);
 Blynk.virtualWrite(V1, h);
 Serial.print("Temperature: ");
 Serial.print(t);
 Serial.print("Humidity: ");
 Serial.println(h);
}
```

OUTPUT:



Conclusion:

The Smart City Air Quality Network (SCAN) represents a visionary and essential initiative for our urbanized world. With the relentless growth of cities and the ever-increasing impact of pollution on public health and the environment, SCAN emerges as a beacon of hope and innovation. By leveraging advanced sensor technology, real-time monitoring, and data-driven insights, SCAN empowers cities to understand, address, and improve air quality comprehensively. This network's significance extends far beyond simply measuring pollutants; it embodies a commitment to public health, environmental preservation, and sustainable urban development.