Air Pollution Monitoring System

Submitted By

Meerab Irfan	**20021519-009**
Hassan Tahir	**20021519-083**
Sheraz Ahmed	**20021519-100**

Supervised By

Dr. Abdur Rehman

BS (Computer Science)

Session 2020 – 2024

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF GUJRAT

Abstract

This proposal introduces "Air Pollution Monitoring System," a comprehensive project designed to monitor various environmental parameters such as temperature, humidity, dust, LPG, & Carbon Monoxide. The system combines cutting-edge sensor technology and data processing to provide real-time insights for environmental conditions.

TABLE OF CONTENTS

Second Deliverable for Structured Approach

Chapter 1 Project Feasibility Report

1.1. Introduction

This proposal aims to develop and present a final year project outlining a comprehensive system for real-time monitoring of environmental parameters. It defines functional and non-functional requirements, scopes the project's timeline, and highlights the methods, tools, and platforms to be utilized.

1.2. Project Title

Air Pollution Monitoring System

1.3. Project Overview Statement

The advancement of smart technologies is crucial in addressing environmental concerns. The proposed "Air Pollution Monitoring System" aims to offer real-time monitoring for parameters like temperature, humidity, dust, LPG, Natural Gas, Ammonia, Smoke & Carbon Monoxide. By integrating advanced sensors, and user-friendly interfaces, the project seeks to provide insights into conditions for various applications. This proposal contextualizes the project, introduces core modules, discusses potential advantages and drawbacks, and outlines pathways for future advancements.

1.4. Target Audience

"Air Pollution Monitoring System" caters to a diverse audience including urban planners, environmental agencies, and individuals interested in maintaining optimal conditions for health, planning, and decision-making.

1.5. Goals & Objectives

Goals:

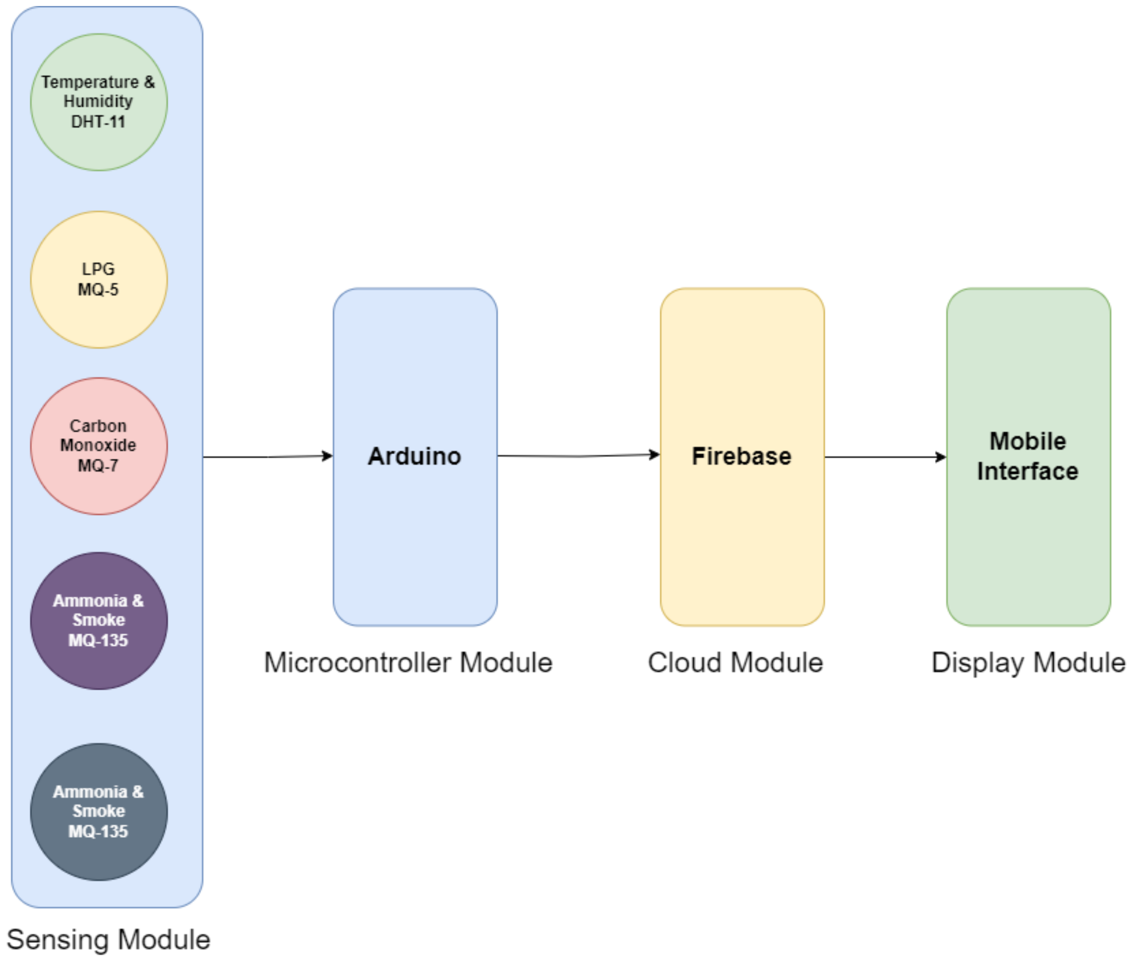
- Develop a comprehensive environment monitoring and forecasting system.
- Provide accurate real-time data on environmental parameters.

Objectives:

- Integrate sensor hardware for seamless data collection.
- Process and preprocess sensor data to derive meaningful insights.
- Design an intuitive interface for data visualization and user interaction.

1.6. Application Architecture

The system requires sensors for temperature, humidity, dust, LPG, Natural Gas, Ammonia, Smoke & Carbon Monoxide and an Arduino and a Wi-Fi module to read input data from sensors and push it online. The Flutter framework will be used for mobile application development. The system will operate on an Arduino platform.



System Architecture

1.7. Hardware and Software Specification

Hardware:

- MQ5
- MQ7
- MQ135
- DHT11
- GP2Y101AU0F
- Arduino Uno
- ESP8266 Wi-Fi Module

Software:

- Arduino IDE
- Flutter
- Firebase

1.8. Estimated Cost

- IoT sensors: PKR 15,000
- Development hardware and software: PKR 5,000
- Total Estimated Cost: PKR 20,000

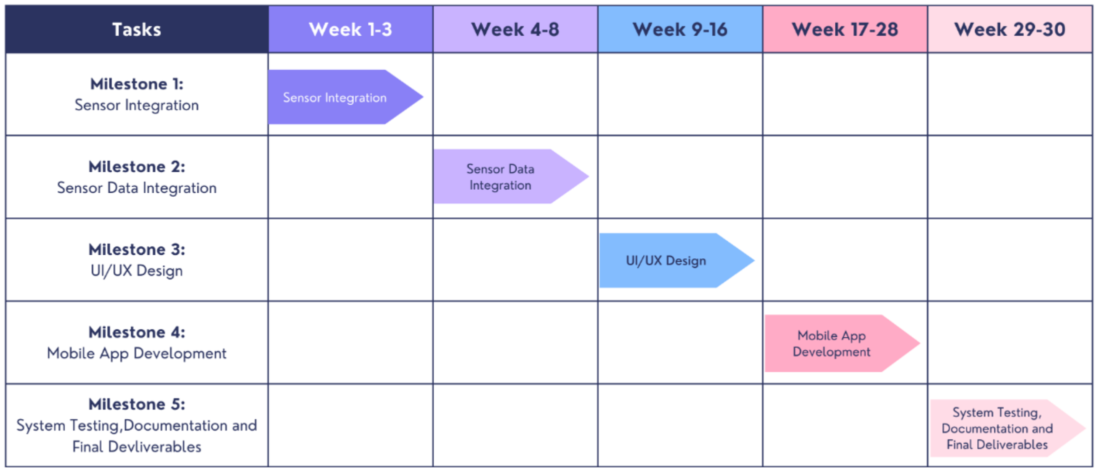
1.9. Tools and Technologies

- Arduino for data collection.
- Wi-Fi Module to push data online.
- Various sensors for temperature, humidity, dust, LPG, Natural Gas, Ammonia, Smoke & Carbon Monoxide.
- Flutter framework for mobile application development.
- Git for version control and collaboration.

1.10. Project Milestones and Deliverables

- Sensor Integration
- Sensor Calibration
- Sensor Data Integration
- Mobile App UI/UX Design
- Mobile App Development
- System testing, documentation, and final deliverables.

Gantt Chart:



Tasks	Week 1-3	Week 4-8	Week 9-16	Week 17-28	Week 29-30
Milestone 1: Sensor Integration	Sensor Integration				
Milestone 2: Sensor Data Integration		Sensor Data Integration			
Milestone 3: UI/UX Design			UI/UX Design		
Milestone 4: Mobile App Development				Mobile App Development	
Milestone 5: System Testing,Documentation and Final Devliverables					System Testing, Documentation and Final Deliverables

1.11. Work division among Group members

The division of responsibilities among the team members is outlined as follows:

- **Sheraz Ahmed:**
  - Role: Sensor Integration
  - Integrate sensors into the system.
  - Ensure proper hardware connectivity and calibration.
- **Meerab Irfan:**
  - Role: UI/UX Design and Project Coordination
  - Design the overall UI/UX of the mobile application.
  - Create visually appealing and intuitive user interfaces.
  - Work on user experience enhancements.
  - Coordinate overall project activities.

- Ensure timely progress and task completion.
- **Hassan Tahir:**
    - Role: Mobile Application Development
    - Develop the mobile application interface.
    - Develop user-friendly controls for user interaction.
    - Integrate Sensor Data into the mobile application.
    - Ensure seamless communication between the app and data processing modules.
    - Optimize the mobile application for performance and responsiveness.

By distributing responsibilities based on strengths and skills, the team aims to ensure efficient progress and successful completion of the "Air Pollution Monitoring System" project. Collaboration among team members will be maintained for a seamless integration of all components.

Second Deliverable for Structured Approach

Chapter 2 Design Document

2.1. Introduction

In today's interconnected world, the impact of environmental factors on human health, ecosystems, and economic stability cannot be understated. Among these factors, air quality remains a significant concern globally, affecting public health and contributing to environmental degradation. Various studies have highlighted the adverse effects of air pollutants such as lead, nitrogen dioxide, benzene, sulfur dioxide, ozone, and particulate matter on human health.

Recognizing the critical role of air quality in health, climate, and ecosystems, numerous regulations have been established worldwide to monitor and control air pollution. However, traditional monitoring stations provide limited real-time data and often publish air quality datasets monthly, creating gaps in continuous monitoring.

In response to the need for continuous and dynamic air quality monitoring, there is a need for a real-time information through an Information System for Continuous Monitoring.

To address these challenges and enhance air quality monitoring, we develop an innovative solution based on the Internet of Things (IoT).

The proposed air quality monitoring tool leverages the Arduino Uno microcontroller coupled with a range of pollutant sensors such as Dust sensor GP2Y101AU0F, MQ5, MQ7, MQ135, and DHT11. These sensors, integrated into the IoT framework, aim to continuously track and relay air pollution emissions data to a mobile application accessible via smartphones.

This project seeks to bridge the gap in real-time air quality monitoring by harnessing IoT technology. By developing a cost-effective, scalable, and accessible solution, the tool aims to not only offer continuous monitoring capabilities but also increase public awareness regarding the importance of maintaining air quality.

2.2 Systems Specifications

The following are the clauses that must be included when describing the system specifications.

2.2.1. Identifying External Entities

- Users: the people who will be using the application to monitor the environment.

- Organization/ Stakeholders: Individuals or groups interested in accessing air quality information. They interact with the system to view air quality data, reports, or receive alerts about pollution levels.
- Sensor Devices: These are external devices that measure air quality parameters such as particulate matter, gases (like CO2, SO2, NOx), temperature, humidity, etc. They provide real-time data to the system.
- Mobile Applications: these interfaces act as external entities for users to interact with the system.

2.2.2. Entity Relationship Diagram

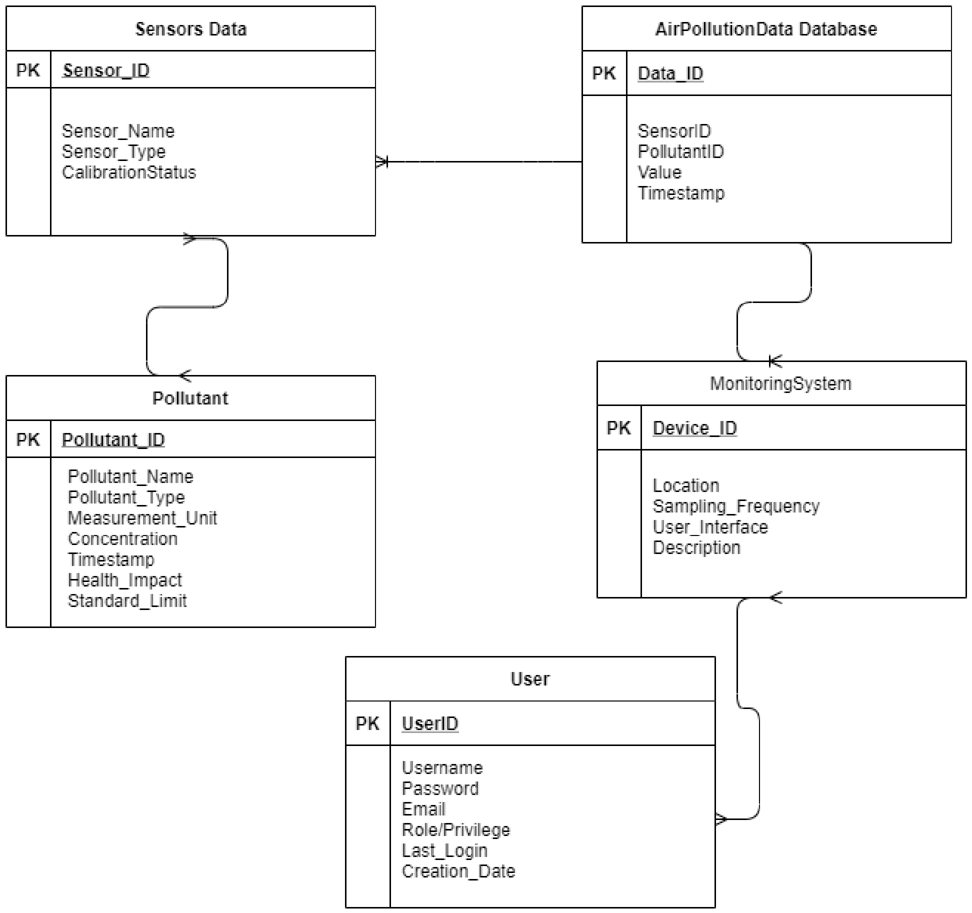
Entities and Attributes:
Monitoring Device:
Attributes: DeviceID (Primary Key), Location, Sampling_Frequency, User_Interface, Description

Sensors:
Attributes: SensorID (Primary Key), Type, CalibrationStatus

Pollutants:
Attributes: PollutantID (Primary Key), Pollutant_Name, Pollutant_Type, Measurement_Unit, Concentration, Timestamp, Health_Impact, Standard_Limit

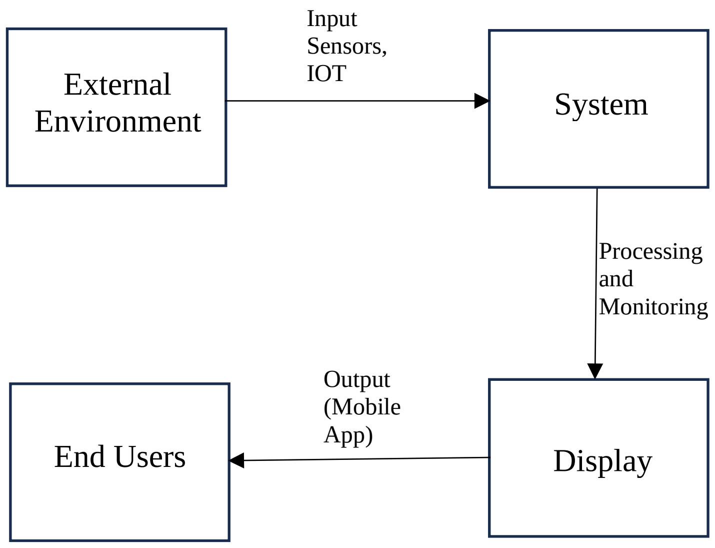
AirPollutionData:
Attributes: Data_ID (Primary Key), SensorID (Foreign Key), PollutantID (Foreign Key), Value, Timestamp

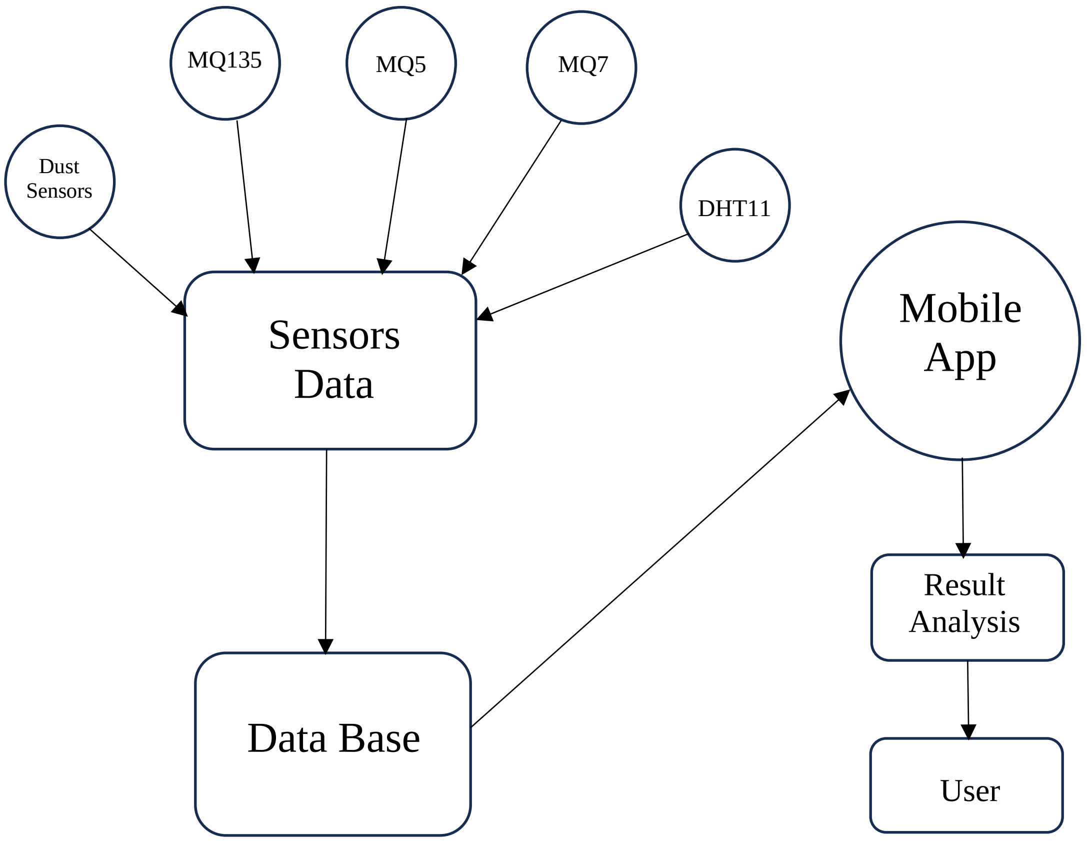
User:
Attributes: Username, Password, Email, Role/Privilege, Last_Login, Creation_Date



Entity Relationship Diagram for “Air Pollution Monitoring System”

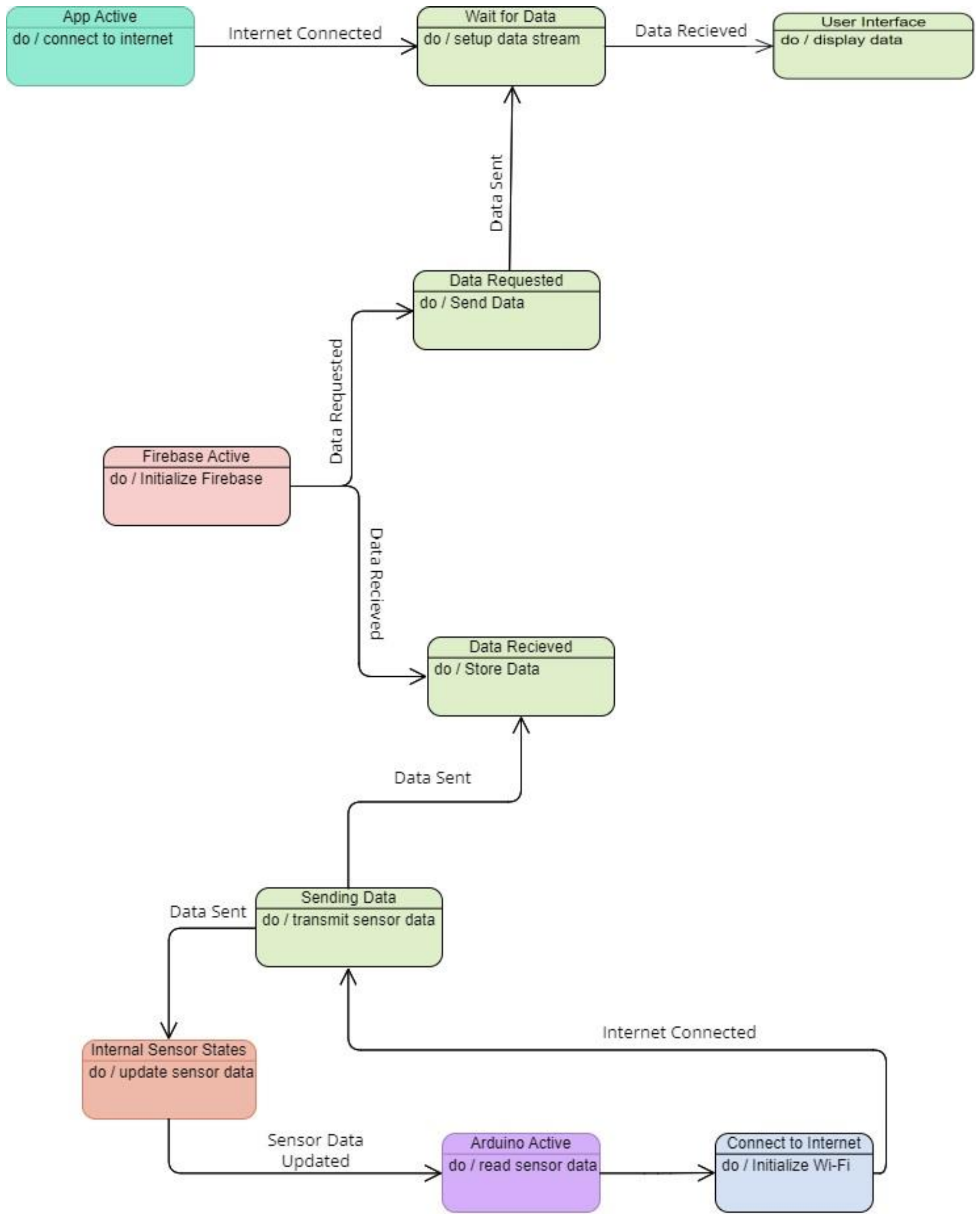
2.2.3. Data flow diagram (Functional Model)





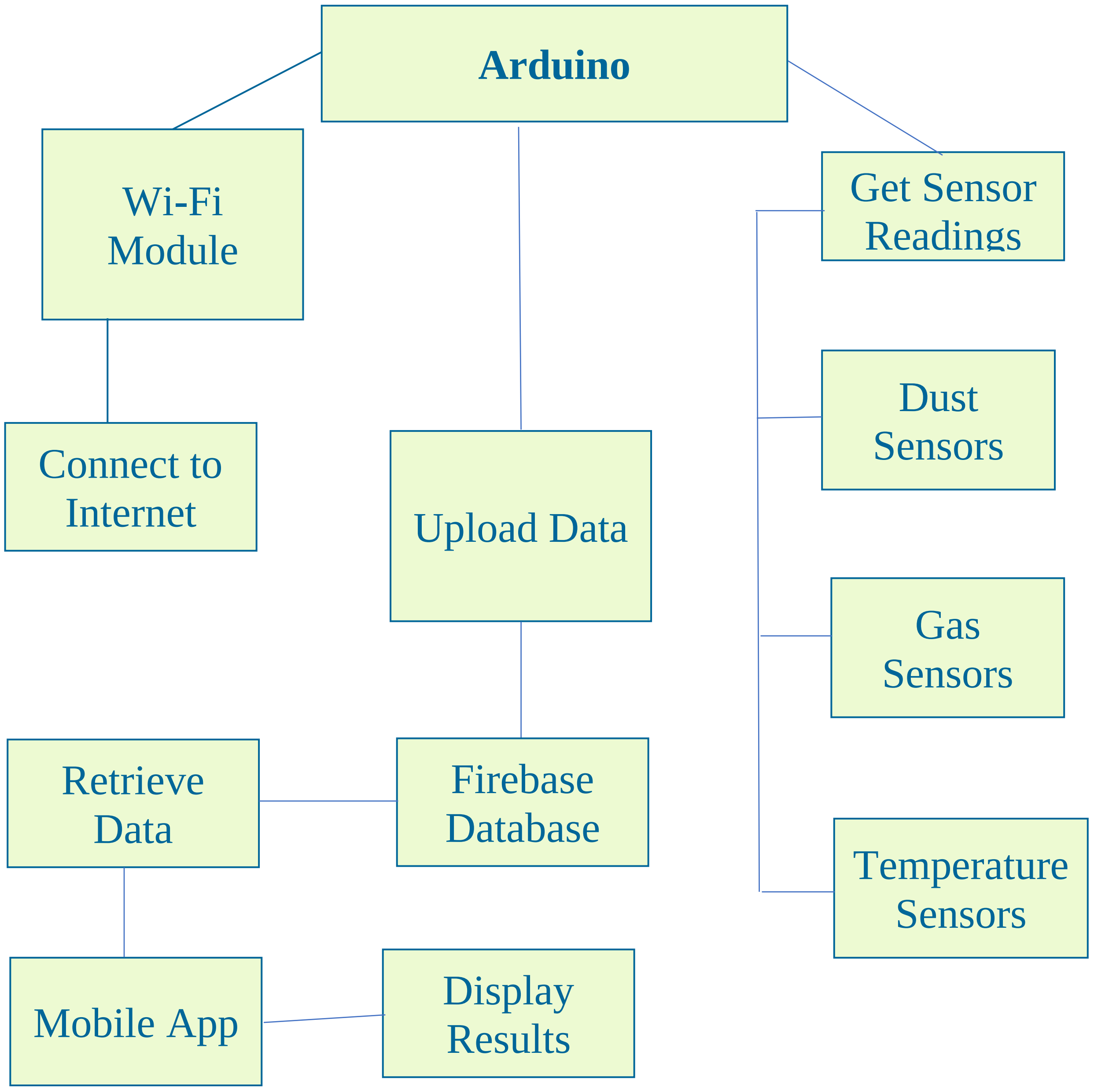
Data Flow Diagram for "Air Pollution Monitoring System"

2.4. State Transition Diagram



State Transition Diagram for “Air Pollution Monitoring System”

2.5. Architectural design



Design Architecture

2.6. Component Level Design

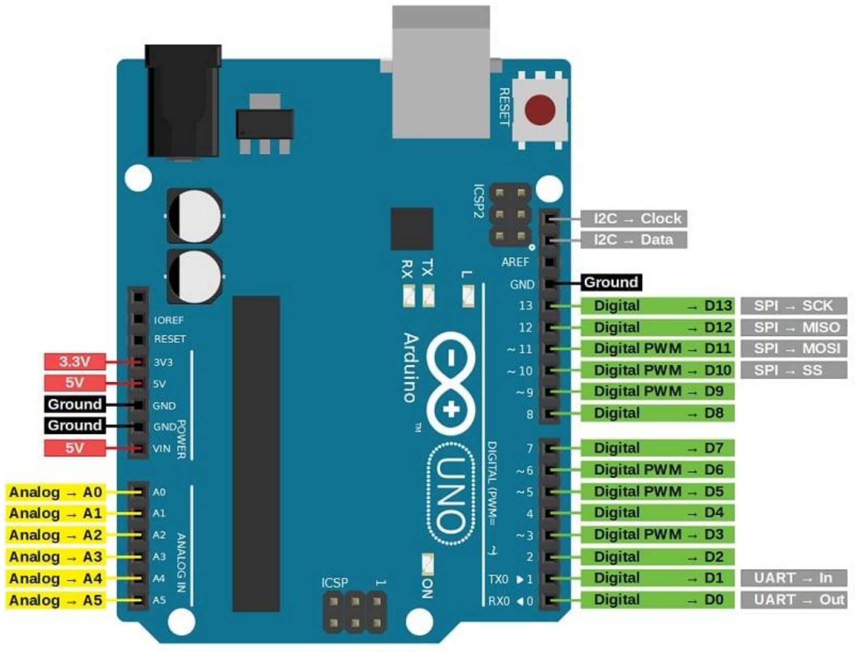
The following are the list of components that we have used in our project:

- **Hardware Components**
    - MQ5 Sensor
    - MQ7 Sensor
    - MQ135 Sensor
    - DHT11 Sensor
    - GP2Y101AU0F Sensor
    - Arduino Uno
    - ESP8266 Wi-Fi Module
    - Breadboard
    - Jumper Wires
- **Software Components**
    - Arduino IDE
    - Flutter
    - Firebase

Arduino Uno

The Arduino Uno is a popular open-source microcontroller board that is part of the Arduino platform. It features an ATmega328P microcontroller, digital and analog input/output pins, USB connectivity, and a versatile development environment. Designed for ease of use, the Arduino Uno is widely utilized by hobbyists, students, and professionals in various projects, including robotics, home automation, and prototyping. Its simplicity, extensive community support, and a vast collection of available shields and modules make the Arduino Uno an excellent choice for individuals entering the world of electronics and programming.

The Arduino Uno has a total of 14 digital pins, 6 analog input pins, and various other pins for power supply and communication.

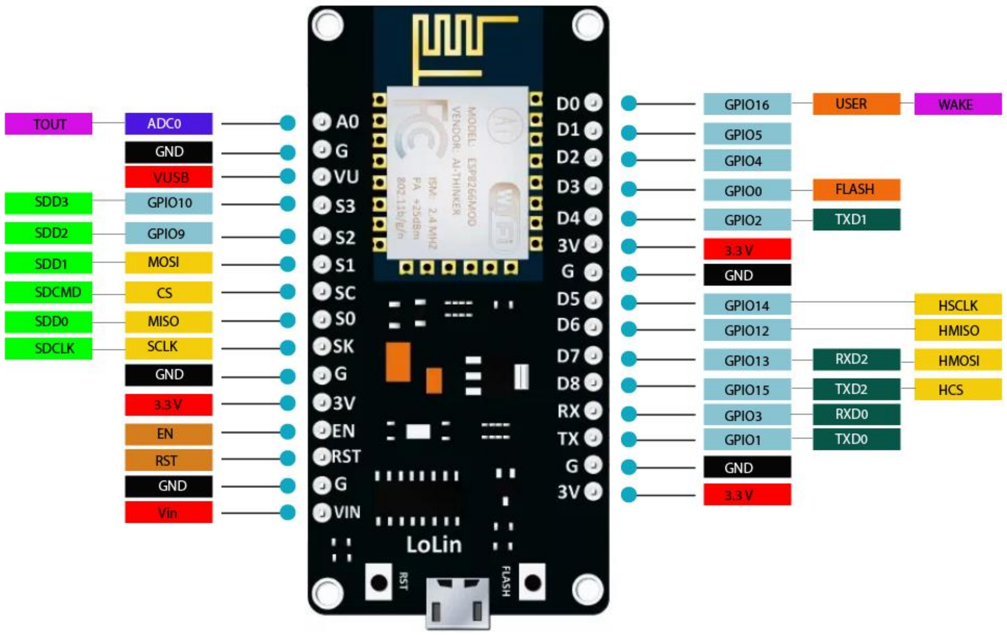


Pinout Diagram of Arduino Uno Microcontroller

- Digital Pins (D2-D13): These pins can be configured as either input or output and are used for digital communication.
- Analog Pins (A0-A5): These pins can be used for analog input, allowing the board to read voltage levels between 0 and 5 volts.
- Power Pins:
  - Vin: Voltage supplied to the board. Can be from an external source or USB.
  - 5V: Regulated 5-volt supply used to power external components.
  - 3.3V: Regulated 3.3-volt supply.
  - GND (Ground): Ground pins for power reference.
- Other Pins:
  - RESET: Used to reset the microcontroller.
  - TX/RX: Serial communication pins (TX for transmission, RX for reception).
  - ARef: Analog reference voltage.
- Communication:
  - ICSP (In-Circuit Serial Programming): Used for programming the microcontroller.
  - USB: Used for connecting the Arduino to a computer for programming and power.

ESP8266 Wi-Fi Module

The ESP8266 is a popular and versatile WiFi module that integrates a microcontroller with built-in WiFi capabilities. It is widely used in IoT (Internet of Things) projects and allows devices to connect to a wireless network and communicate with other devices or the internet. The ESP8266 modules come in various versions, but one of the most common is the ESP-01.



Pinout Diagram of ESP8266 Wi-Fi Module

The ESP-01 module has 8 pins, but the most used pins are:

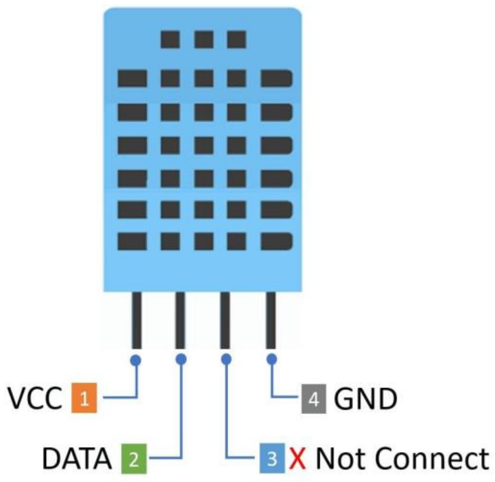
- VCC: Power supply pin. Connect this to a 3.3V power source.
- GND: Ground pin.
- TX (Transmit): Transmit data pin. Connect this to the RX pin on the device you are communicating with (e.g., Arduino).
- RX (Receive): Receive data pin. Connect this to the TX pin on the device you are communicating with.
- GPIO0 (General Purpose Input/Output 0): General-purpose digital input/output pin. Its state during boot determines the ESP8266 operating mode.
- RST (Reset): Reset pin. Pulled low to reset the module.
- CH_PD (Chip Power-Down): Chip power-down pin. Connect this to VCC for normal operation.
- GPIO2 (General Purpose Input/Output 2): General-purpose digital input/output pin.

Pseudocode for connection:

```
function setup():
    initializeSerial()
    connectToWiFi("YourSSID", "YourPassword")

function loop():
    if (isWiFiConnected()):
        // Perform tasks when connected to Wi-Fi
        // Example: sendSensorDataToServer()
    else:
        // Reconnect to Wi-Fi if disconnected
        reconnectToWiFi()

function connectToWiFi(ssid, password):
    // Use ESP8266 Wi-Fi library to connect to Wi-Fi network
    WiFi.begin(ssid, password)

    // Wait for Wi-Fi connection
    while (WiFi.status() != WL_CONNECTED):
        delay(1000)

    print("Connected to Wi-Fi")

function isWiFiConnected():
    // Check if the ESP8266 is currently connected to Wi-Fi
    return (WiFi.status() == WL_CONNECTED)

function reconnectToWiFi():
    // Attempt to reconnect to Wi-Fi
    connectToWiFi("YourSSID", "YourPassword")
```

DHT11 Sensor

The DHT11 is a low-cost digital temperature and humidity sensor module commonly used in electronic projects. It includes a calibrated digital signal output that provides accurate readings of temperature and humidity. With a single-wire digital interface, the DHT11 is easy to use with microcontrollers like Arduino, Raspberry Pi, and other platforms. This sensor is popular in applications such as weather stations, home automation, and environmental monitoring, providing a cost-effective solution for measuring ambient conditions.



Pinout Diagram of DHT11 Sensor

The DHT11 sensor typically has four pins:

- VCC: Power supply pin. Connect this to the 5V pin on the Arduino.
- Data (DOUT or OUT): This pin is used for both sending data from the sensor to the Arduino and receiving commands or initialization from the Arduino. Connect this to a digital pin on the Arduino.
- Not Connected (NC): This pin is often labeled NC and is not used. It can be left unconnected.
- Ground (GND): Connect this pin to the ground (GND) pin on the Arduino.

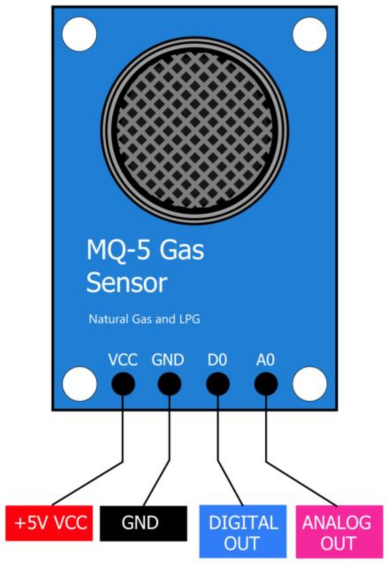
Pseudocode for calibration of DHT11 is given below:

```
DHT_PIN = N
DHT_TYPE = DHT11

dht = initializeDHT(DHT_PIN, DHT_TYPE)

function setup():
   initializeSerial()
   dht.begin()
```

```
function loop():
    delay(2000)
    humidity = dht.readHumidity()
    temperature = dht.readTemperature()

    if (isNaN(humidity) or isNaN(temperature)):
        print("Failed to read from DHT sensor!")
        return

    temperature = temperature + 2.0
    humidity = humidity - 5.0

    print("Temperature: " + temperature + " °C")
    print("Humidity: " + humidity + " %")
```

MQ5 Sensor

The MQ-5 sensor is a gas sensor that detects the presence of various flammable gases, such as methane, propane, butane, and LPG (liquefied petroleum gas). It operates on the principle of semiconductor conductivity, with its resistance changing in the presence of target gases. Widely used in gas leakage detection systems and industrial applications, the MQ-5 sensor provides a cost-effective and reliable solution for monitoring combustible gas concentrations in the environment.



Pinout Diagram of MQ5 Sensor

The MQ-5 sensor typically has four pins:

- AO (Analog Output): Analog voltage output that varies with the concentration of the detected gas. Connect this pin to an analog input pin on the Arduino if you want to read analog values.
- DO (Digital Output): Digital output that provides a HIGH or LOW signal depending on whether the concentration of gas exceeds a predefined threshold. Connect this pin to a digital input pin on the Arduino.
- GND (Ground): Connect this pin to the ground (GND) on the Arduino.
- VCC (Voltage Common Collector): Power supply pin. Connect this to the 5V pin on the Arduino.

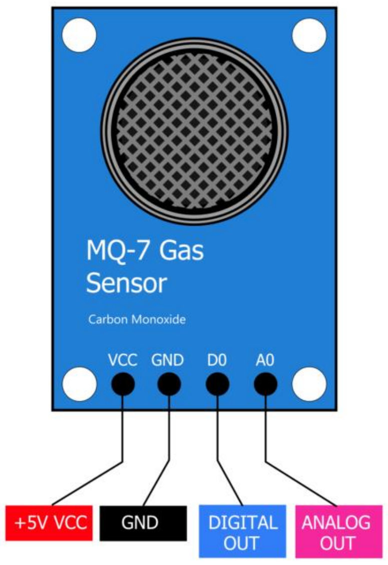
Pseudocode for calibration of MQ5 Sensor:

```
MQ_PIN = AnalogPin(N)  // Analog pin where the MQ-5 sensor is connected

function setup():
   initializeSerial()

function loop():
   // Read analog value from MQ-5 sensor
   rawValue = analogRead(MQ_PIN)

   // Convert analog value to voltage
```

```
    voltage = rawValue * (5.0 / 1023.0)

    // Perform gas concentration calculation (calibration may be needed)
    gasConcentration = calculateGasConcentration(voltage)

    // Print the gas concentration
    print("Gas Concentration: " + gasConcentration + " ppm")
    delay(2000)  // Wait for 2 seconds before the next reading

function calculateGasConcentration(voltage):
    // Calibration parameters (adjust as needed)
    VOLTAGE_CLEAN_AIR = 1.0  // Voltage in clean air
    RS_CLEAN_AIR = 5.0      // Sensor resistance in clean air

    // Calculate sensor resistance
    RS = (5.0 - voltage) / voltage

    // Calculate gas concentration using a simple formula (calibration may be needed)
    gasConcentration = (RS / RS_CLEAN_AIR) * VOLTAGE_CLEAN_AIR

    return gasConcentration
```

MQ7 Sensor

The MQ-7 sensor is a gas sensor designed to detect the presence of carbon monoxide (CO) and natural gas in the air. Operating on the principle of semiconductor conductivity, the MQ-7 sensor's resistance changes in the presence of these gases. It is commonly used in applications where monitoring indoor air quality and detecting potentially harmful gas leaks are essential, making it a crucial component in gas detection systems for homes and industrial settings.



Pinout Diagram of MQ7 Sensor
The MQ-7 sensor typically has four pins:

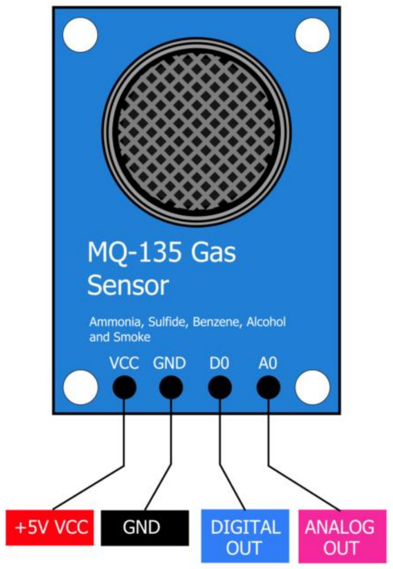
- A (Analog Output): Analog voltage output that varies with the concentration of the detected gas. Connect this pin to an analog input pin on the Arduino if you want to read analog values.
- D (Digital Output): Digital output that provides a HIGH or LOW signal depending on whether the concentration of gas exceeds a predefined threshold. Connect this pin to a digital input pin on the Arduino.
- GND (Ground): Connect this pin to the ground (GND) on the Arduino.
- VCC (Voltage Common Collector): Power supply pin. Connect this to the 5V pin on the Arduino.

Pseudocode for calibration of MQ7 Sensor:

```
MQ_PIN = AnalogPin(N)  // Analog pin where the MQ-7 sensor is connected

function setup():
    initializeSerial()

function loop():
    // Read analog value from MQ-7 sensor
    rawValue = analogRead(MQ_PIN)

    // Convert analog value to voltage
    voltage = rawValue * (5.0 / 1023.0)

    // Perform gas concentration calculation (calibration may be needed)
    gasConcentration = calculateGasConcentration(voltage)

    // Print the gas concentration
    print("Gas Concentration: " + gasConcentration + " ppm")
    delay(2000)  // Wait for 2 seconds before the next reading

function calculateGasConcentration(voltage):
    // Calibration parameters (adjust as needed)
    VOLTAGE_CLEAN_AIR = 1.0  // Voltage in clean air
    RS_CLEAN_AIR = 5.0      // Sensor resistance in clean air

    // Calculate sensor resistance
    RS = (5.0 - voltage) / voltage

    // Calculate gas concentration using a simple formula (calibration may be needed)
    gasConcentration = (RS / RS_CLEAN_AIR) * VOLTAGE_CLEAN_AIR

    return gasConcentration
```

MQ135 Sensor

The MQ-135 is a gas sensor designed to detect a variety of gases, including ammonia (NH_3), methane (CH_4), carbon dioxide (CO_2), and other volatile organic compounds (VOCs). It is commonly used for air quality monitoring in indoor environments. The sensor operates on the principle of a semiconductor and changes its resistance in the presence of the target gases.



Pinout Diagram of MQ135 Sensor

The MQ-135 sensor typically has four pins:

- A (Analog Output): Analog voltage output that varies with the concentration of the detected gas. Connect this pin to an analog input pin on the Arduino if you want to read analog values.
- D (Digital Output): Digital output that provides a HIGH or LOW signal depending on whether the concentration of gas exceeds a predefined threshold. Connect this pin to a digital input pin on the Arduino.
- GND (Ground): Connect this pin to the ground (GND) on the Arduino.
- VCC (Voltage Common Collector): Power supply pin. Connect this to the 5V pin on the Arduino.

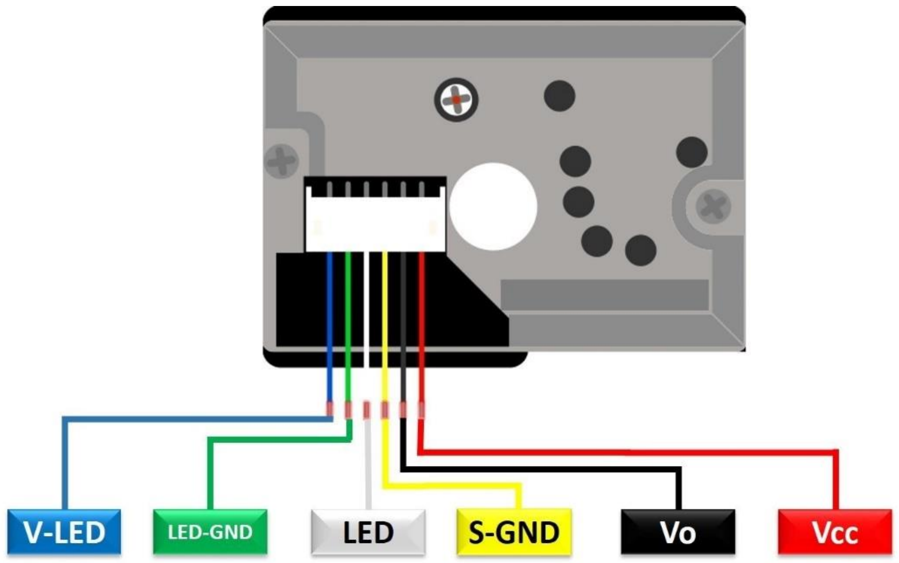
Pseudocode for calibration:

```
MQ_PIN = AnalogPin(N)  // Analog pin where the MQ-135 sensor is connected

function setup():
   initializeSerial()

function loop():
   // Read analog value from MQ-135 sensor
   rawValue = analogRead(MQ_PIN)

   // Convert analog value to voltage
```

```
    voltage = rawValue * (5.0 / 1023.0)

    // Perform gas concentration calculation (calibration may be needed)
    gasConcentration = calculateGasConcentration(voltage)

    // Print the gas concentration
    print("Gas Concentration: " + gasConcentration + " ppm")
    delay(2000)  // Wait for 2 seconds before the next reading

function calculateGasConcentration(voltage):
    // Calibration parameters (adjust as needed)
    VOLTAGE_CLEAN_AIR = 1.0  // Voltage in clean air
    RS_CLEAN_AIR = 5.0      // Sensor resistance in clean air

    // Calculate sensor resistance
    RS = (5.0 - voltage) / voltage

    // Calculate gas concentration using a simple formula (calibration may be needed)
    gasConcentration = (RS / RS_CLEAN_AIR) * VOLTAGE_CLEAN_AIR

    return gasConcentration
```

GP2Y101AU0F Sensor

The GP2Y101AU0F is an optical sensor designed for detecting fine particles (dust) in the air. It utilizes an infrared LED and a photodetector to measure the scattering of light caused by airborne particles. This sensor is commonly used in air quality monitoring applications to assess the concentration of particulate matter.



Pinout Diagram of GP2Y101AU0F

The GP2Y101AU0F sensor typically has six pins:

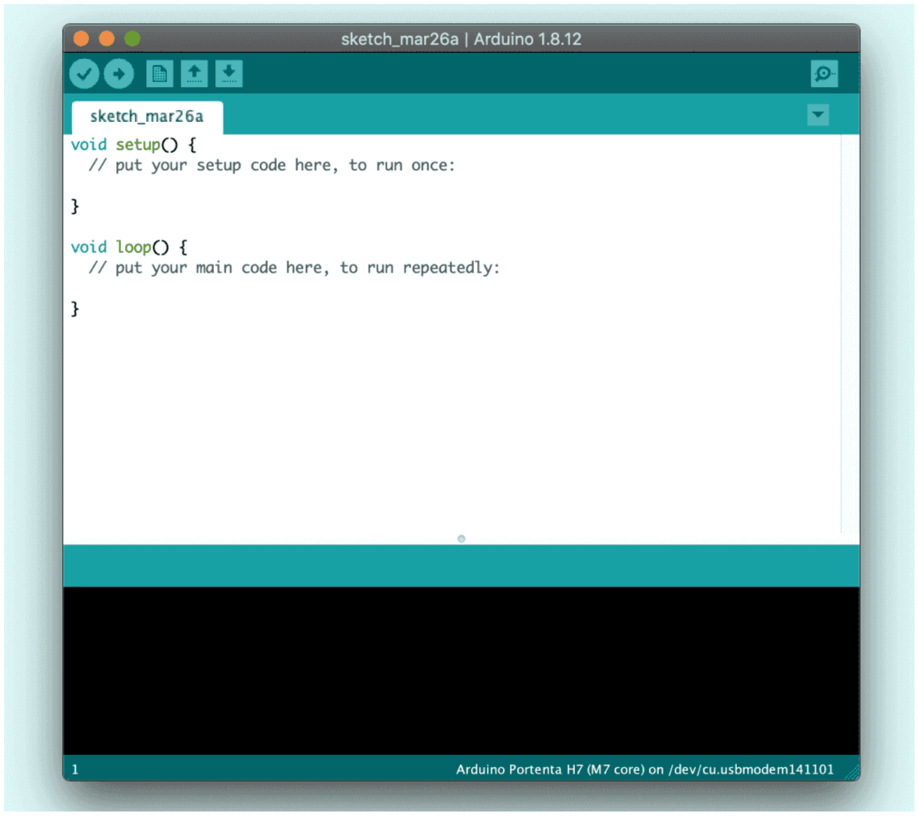
- V-OUT (Analog Output): Analog voltage output that varies with the concentration of dust particles. Connect this pin to an analog input pin on the Arduino if you want to read analog values.
- LED: This pin is connected to an infrared LED that emits light for particle detection. Connect this pin to a digital output pin on the Arduino to control the LED.
- GND (Ground): Connect this pin to the ground (GND) on the Arduino.
- Vo (Output Voltage): This pin provides the output voltage directly related to the dust density. It is an alternative analog output pin and can be connected to an analog input on the Arduino.
- Vcc (Power Supply): Power supply pin. Connect this to the 5V pin on the Arduino.
- LED Ground: This pin is the ground connection for the infrared LED. Connect it to the ground (GND) on the Arduino.

Pseudocode for calibration:

```
GP2Y_PIN = AnalogPin(N)

function setup():
   initializeSerial()

function loop():
   // Read analog value from GP2Y101AU0F sensor
   rawValue = analogRead(GP2Y_PIN)

   // Convert analog value to voltage
   voltage = rawValue * (5.0 / 1023.0)

   // Perform particle concentration calculation (calibration may be needed)
   particleConcentration = calculateParticleConcentration(voltage)

   // Print the particle concentration
   print("Particle Concentration: " + particleConcentration + " pcs/0.01cf")
   delay(2000)  // Wait for 2 seconds before the next reading

function calculateParticleConcentration(voltage):
   // Calibration parameters (adjust as needed)
   VOLTAGE_CLEAN_AIR = 0.6  // Voltage in clean air
   K = 0.5                  // Sensitivity factor (adjust as needed)

   // Calculate particle concentration using a simple formula (calibration may be needed)
   particleConcentration = K * (voltage - VOLTAGE_CLEAN_AIR)

   return particleConcentration
```

Arduino IDE

The Arduino IDE is a cross-platform software application that provides a comprehensive environment for programming and uploading code to Arduino microcontroller boards. It simplifies the process of writing, compiling, and uploading code to the Arduino hardware, making it accessible to both beginners and experienced developers. The IDE includes a code editor, a compiler, and a bootloader for uploading code to the Arduino board. It supports the Arduino programming language, which is a simplified version of C and C++. The Arduino IDE also integrates with various libraries and allows users to manage their projects easily. Overall, it serves as a user-friendly tool for developing and uploading code to Arduino-based projects.


sketch_mar26a | Arduino 1.8.12
sketch_mar26a
1
Arduino Portenta H7 (M7 core) on /dev/cu.usbmodem141101

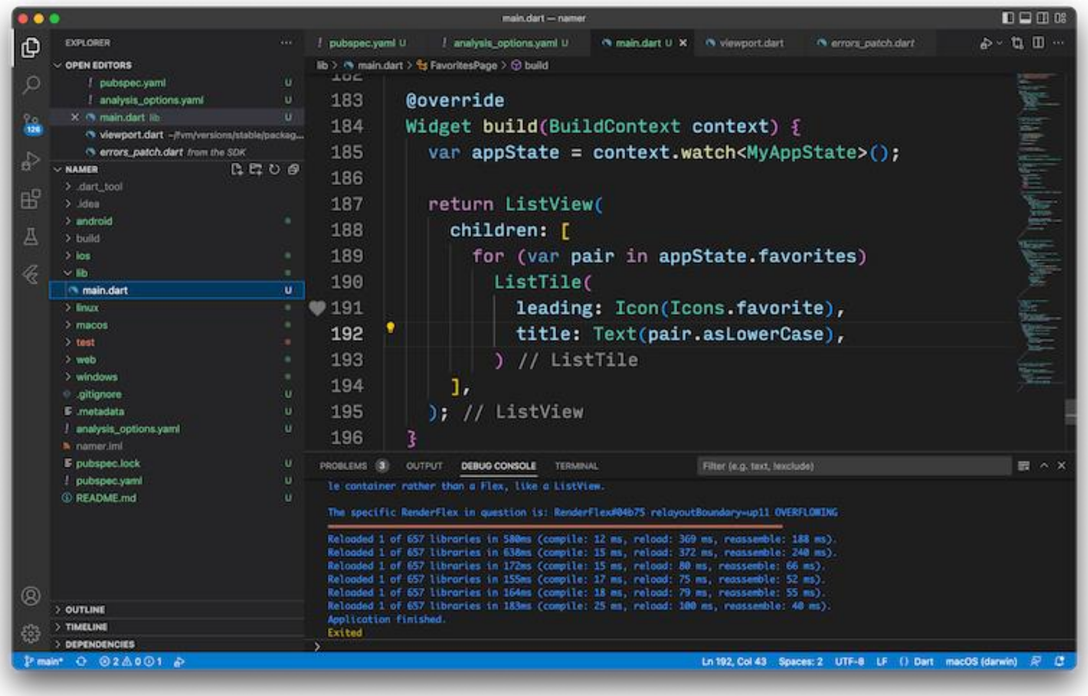
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Arduino IDE

The program or code written in the Arduino IDE is often called sketching. We need to connect the Genuino and Arduino board with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino.'

Flutter

Flutter is an open-source UI software development toolkit created by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and provides a rich set of pre-designed widgets that help developers create visually appealing and consistent user interfaces across different platforms. Flutter's "hot reload" feature allows for quick and efficient development, enabling developers to see changes instantly during the development process.



```
  @override
  Widget build(BuildContext context) {
    var appState = context.watch<MyAppState>();

    return ListView(
      children: [
        for (var pair in appState.favorites)
          ListTile(
            leading: Icon(Icons.favorite),
            title: Text(pair.asLowerCase),
          ), // ListTile
      ],
    ); // ListView
  }
```

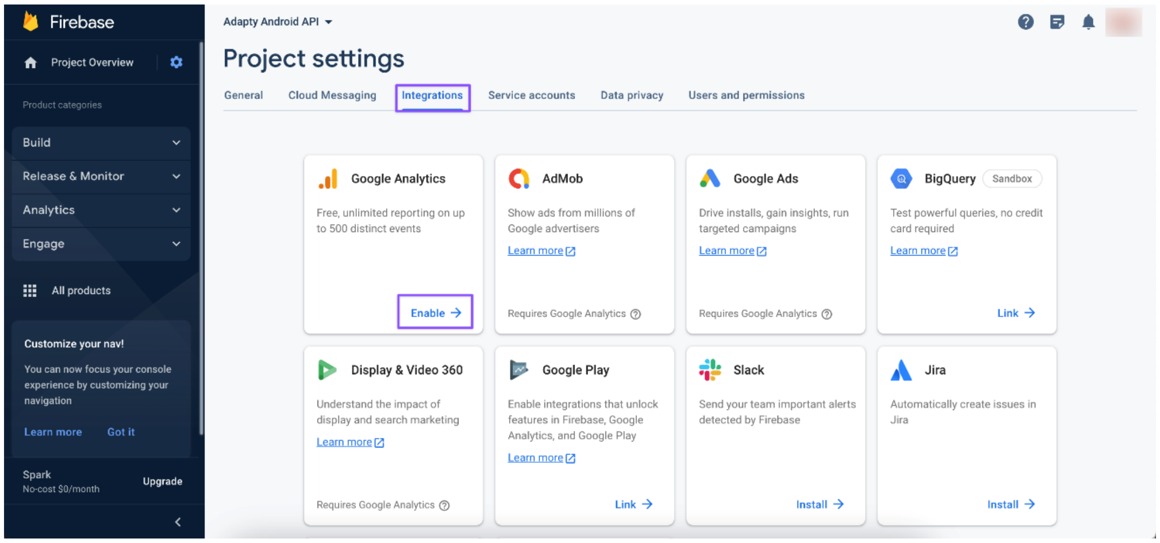
```
le container rather than a Flex, like a ListView.

The specific RenderFlex in question is: RenderFlex#04b75 relayoutBoundary=up11 OVERFLOWING

Reloaded 1 of 657 libraries in 580ms (compile: 12 ms, reload: 369 ms, reassemble: 188 ms).
Reloaded 1 of 657 libraries in 638ms (compile: 15 ms, reload: 372 ms, reassemble: 240 ms).
Reloaded 1 of 657 libraries in 172ms (compile: 15 ms, reload: 80 ms, reassemble: 66 ms).
Reloaded 1 of 657 libraries in 155ms (compile: 17 ms, reload: 75 ms, reassemble: 52 ms).
Reloaded 1 of 657 libraries in 164ms (compile: 18 ms, reload: 79 ms, reassemble: 55 ms).
Reloaded 1 of 657 libraries in 183ms (compile: 25 ms, reload: 100 ms, reassemble: 40 ms).
Application finished.
Exited
```

Flutter running on VS Code

It has gained popularity for its expressive and flexible design, high-performance rendering engine, and the ability to create beautiful and responsive applications across various platforms using a single codebase.

Firebase

Firebase is a comprehensive mobile and web application development platform offered by Google. It provides a variety of tools and services that facilitate the development process, enabling developers to build scalable, feature-rich applications more efficiently.



Project settings

Customize your nav!

You can now focus your console experience by customizing your navigation

Google Analytics — Free, unlimited reporting on up to 500 distinct events

AdMob — Show ads from millions of Google advertisers — Requires Google Analytics

Google Ads — Drive installs, gain insights, run targeted campaigns — Requires Google Analytics

BigQuery — Test powerful queries, no credit card required

Display & Video 360 — Understand the impact of display and search marketing — Requires Google Analytics

Google Play — Enable integrations that unlock features in Firebase, Google Analytics, and Google Play

Slack — Send your team important alerts detected by Firebase

Jira — Automatically create issues in Jira

Main Dashboard of Firebase

Key components of Firebase include:

- **Realtime Database:** A cloud-hosted NoSQL database that allows developers to store and sync data in real-time across multiple clients.
- **Authentication:** Firebase Authentication provides a secure and easy-to-implement user authentication system, supporting email/password, social media logins, and more.
- **Cloud Firestore:** A NoSQL document database for web and mobile applications that offers seamless data synchronization and real-time updates.
- **Cloud Functions:** Serverless functions that can be triggered by events in your Firebase features or HTTP requests, allowing for server-side logic without managing infrastructure.
- **Cloud Storage:** A scalable object storage solution for storing and serving user-generated content like images and videos.