A Mini Project Report On

IoT-Based Smart Industrial Safety System

Submitted in Partial fulfillment of the requirements for the award of the Degree of

Bachelor of Technology

In

Department of Computer Science and Engineering

By

Kallem Sahithi	**22241A0587**
Jannu Rashmi Chandana	**22241A0585**
Parupalli Bhavya	**22241A05B2**
Kalewar Sannitha	**22241A0586**

Under the Esteemed guidance of

Dr. D. Siri

Associate Professor

Department of Computer Science and Engineering

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous) Bachupalli, Kukatpally, Hyderabad, Telangana, India,500090

2024-2025

GOKARAJU RANGARAJU

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

CERTIFICATE

This is to certify that the Mini Project entitled **"IoT-Based Smart Industrial Safety System"** is submitted by **Kallem Sahithi (22241A0587), Jannu Rashmi Chandana(22241A0585), Parupalli Bhavya (22241A05B2), Kalewar Sannitha(22241A0586),** Partial fulfillment of the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in Computer Science and Engineering during the academic year 2024-2025.

INTERNAL GUIDE	HEAD OF THE DEPARTMENT
Dr. D. Siri	**Dr. B. SANKARA BABU**
Associate Professor	**Professor & HoD**

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Many people helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First, we wish to express our deep gratitude to our internal guide **Dr. D. Siri, Associate Professor**, Department of CSE for her support in the completion of our project report. We wish to express our honest and sincere thanks to **Mr**. **K. Chandra Mouli and Mr. H. Kalyan** for coordinating in conducting the project reviews. We express our gratitude to **Dr. B. Sankara Babu, HOD**, department of CSE for providing resources, and to the principal **Dr. J. Praveen** for providing the facilities to complete our Mini Project**.** We would like to thank all our faculty and friends for their help and constructive criticism during the project completion phase. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

Kallem Sahithi (22241A0587)

Jannu Rashmi Chandana (22241A0585)

Parupalli Bhavya (22241A05B2)

Kalewar Sannitha (22241A0586)

DECLARATION

We hereby declare that the Mini Project entitled **"IOT-Based Smart Industrial Safety System"** is the work done during the period from 16th Jan 2025 to 13th May 2025 and is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering from **Gokaraju Rangaraju Institute of Engineering and Technology.** The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

Kallem Sahithi (22241A0587)

Jannu Rashmi Chandana (22241A0585)

Parupalli Bhavya (22241A05B2)

Kalewar Sannitha (22241A0586)

LIST OF TABLES

ABSTRACT

The industrial security system is an IoT-based future system trying to increase the security of the industrial sector through continuous monitoring and immediate notice. The system focuses on an ESP32 microcontroller and includes various sensors such as temperature and humidity sensors, gas sensors, air quality sensors, and flame sensors to understand potential hazards such as gas leaks, fire outbreaks, and polluted air. The system uses Twilio API to send SMS notifications to fire stations and hospitals.

The main functionality of the system is live monitoring of the most important environmental parameters, which are carefully monitored when it comes to sensory deviations with pre- or post-oriented thresholds. The main parameters are powerful to the Blynk application, which shows them, and Firebase provides logging of real-time for later analysis. One-time events trigger SMS notifications, using flag-based arguments to avoid duplicates and also increase through a Buzzer and led by local visual and hearing notice, which has manual override functionality on Blynk.

With a focus on operational efficiency, the system provides a fixed basis for rapid, accurate identification and response, leading to effective emergency management. It is scalable in design so that it can fit different industrial environments. Increased sensor integration and other sophisticated safety work can be introduced to increase performance in future releases. The project provides a practical application of IoT technology for industrial security with a fixed basis for monitoring the workplace in real time and emergency communication.

CHAPTER 1
INTRODUCTION

Industrialization is one of the most important contributors to both economic growth and the modernization of many countries across the globe. Whether it's manufacturing units and chemical plants or power generation and oil refineries, industries have enabled mass production and played a significant role in the development of the nation. But with the advantages come inevitable risks. Industrial environments typically deal with flammable substances, high temperatures, toxic gases, and heavy machinery, making them vulnerable to fire outbreaks, gas leaks, and smoke hazards. In unfortunate situations, even a minor safety lapse can result in terrible consequences including loss of human life, damage to environment, and economic loss. Therefore, ensuring safety in industries is not just a legal obligation but also a moral and operational necessity.

In the past safety in industry was essentially manual inspection and Analog alarm systems and periodic checks that were often more reactive than proactive. In comparison, the Internet of Things-enabled approach is more versatile and all-encompassing. Although it should be noted that a comprehensive method has its limits. The Internet of Things approach is geared towards real-time monitoring, timely alerts, and efficient data logging. This is because the Internet of Things allows for continuous monitoring of the parameters that are being checked. Also, we can be much more certain that we are fully aware of all the parameters that need to be checked.

In recent years IoT is becoming increasingly common in such sectors as agriculture, healthcare, home automation, and smart cities. One of the most impactful applications of IoT is in industrial safety, as it can change the way potential risks are detected and managed. By integrating IoT-enabled sensors in the industrial environment, it becomes possible to monitor the concentration of gas, temperature, humidity, smoke density, and flame detection in real-time.

Our project, IoT-Based Industrial Safety System with Gas Fire and Smoke Detection, proposes an innovative system to confront industrial safety concerns in a more reliable, extensible, and economical way by employing multiple sensors to detect hazardous gases (e.g. LPG, methane, carbon monoxide), smoke particles (indicative of a fire), and a direct flame or high-temperature

signal connected to an Arduino or ESP32 microcontroller that processes the sensor output and alerts the mechanism via Internet connectivity, setting off local alarms (buzzers and LEDs) and sending notifications through mobile applications or cloud platforms to the relevant personnel on identifying a dangerous situation.

This system has one peculiar feature – real-time reaction. Unlike traditional security systems that can only alarm on the spot, our solution is based on the Internet of Things and sends alarms immediately to remote users – by SMS, email or app notifications. It significantly decreases the response time of authorities or workers to the possible emergency preventing the exacerbation. What is more, the system keeps logging sensor data constantly, allowing the monitoring of environmental trends, risk assessment, and predictive maintenance.

Our project, IoT-Based Industrial Safety System with Gas Fire and Smoke Detection, proposes an innovative system to confront industrial safety concerns in a more reliable, extensible, and economical way by employing multiple sensors to detect hazardous gases (e.g. LPG, methane, carbon monoxide), smoke particles (indicative of a fire), and a direct flame or high-temperature signal connected to an Arduino or ESP32 microcontroller that processes the sensor output and alerts the mechanism via Internet connectivity, setting off local alarms (buzzers and LEDs) and sending notifications through mobile applications or cloud platforms to the relevant personnel on identifying a dangerous situation.

This system has one peculiar feature – real-time reaction. Unlike traditional security systems that can only alarm on the spot, our solution is based on the Internet of Things and sends alarms immediately to remote users – by SMS, email or app notifications. It significantly decreases the response time of authorities or workers to the possible emergency preventing the exacerbation. What is more, the system keeps logging sensor data constantly, allowing the monitoring of environmental trends, risk assessment, and predictive maintenance.

In the world, where the industrial accidents have a threat to life day by day, our project provides the practical solution that uses the possibilities of modern technology. This project combines the detection of gas, smoke, and fire into one IoT-platform. We hope to create a tool for industrial companies that can save lives and provide the culture of safety, preparedness, and data-based decisions. Our vision is to make the work safer with the help of smart, cheap, and reliable technology.

CHAPTER 2
LITERATURE SURVEY

Increasing workplace safety is becoming a major problem of our time, especially in the industrial works, where hazardous gases, fire, and smoke can put a human life in danger. That is why the issues of smart monitoring systems are seeing the active development and research. With the growing accessibility of Internet of Things (IoT), a number of studies have proposed different systems that can improve safety by means of real-time detection and alerting mechanisms or remote monitoring. In this survey, we will review some of the major works, which have set the foundation for the development of our project, namely the system of gas, fire, and smoke detection on the basis of IoT, which is oriented to the industrial sector.

[1] In the recent study of "IoT-HGDS: Internet of Things Integrated Machine Learning-Based Hazardous Gases Detection System for Smart Kitchen" (2024), the authors propose on a gas detection system conducted with the help of IoT and machine learning techniques (such as SVM and RF). The system was designed for smart kitchens and sends real-time notifications with the help of buzzers and mobile alerts. Other advantages include cost-effectiveness and cloud-based connectivity. However, the device was tested in a simulated setting, which is why the authors cannot speak about its further application, scalability, or sensor drift. However, the study shows how one can increase the precision of detection and the speed of response by integrating AI and IoT.

[2] "Low-Cost IoT-Based Sensor System: A Case Study on Harsh Environmental Monitoring" (2021) is another meaningful project. It is focused on demonstrating the affordability and energy efficiency. It monitors hydrogen gas, temperature, humidity, and pressure in nuclear waste storage environments with the help of sensors like MQ-8 and BME680, while solar energy provides the power for transmission using Wi-Fi to the ThingSpeak platform. Still, the deficiency of the project is that it can monitor only hydrogen and that it is not shielded for high radiation. Nonetheless, the project demonstrates the importance of creating adaptive, small, and energy-efficient systems for harsh conditions

[3] The essay "IOT Based Intelligent Industry Monitoring System" (2019) directly addresses industrial settings. Smart sensors and Raspberry Pi monitor temperature, humidity, and gases like CO, LPG and methane, uploading the data to Google Cloud. The system generates alerts when the thresholds are breached. It can be embedded in wearable tech for worker safety. However, the system lacks predictive analytics and advanced features like pattern recognition or interaction management among gases.

[4] Likewise, "IoT-Based Industrial Plant Safety Gas Leakage Detection System" (2018) employs MQ6, MQ4, and MQ135 sensors and ESP32 for LPG, methane, and benzoene gas leaks detection. Real-time data are transmitted to the UBIDOTS cloud, and buzzer and LED notification are initiated through IFTTT. The system is easy to use and works effectively but is limited in that it does not have integration with emergency services and is calibration-dependent, potentially impacting long-term accuracy and scalability

[5] In addition, "IoT-Based Industrial Gas Leakage Detection" (2024) focuses on usability and accessibility. With the help of an ESP8266 module and a set of gas sensors, it detects leakage and warns users through SMS and LED indicators (e.g., red for LPG). Simplicity is the focus of the system, but there is no mention of accuracy measures or comparison with commercial solutions. Internet reliance can also be a restricting element in far-off locations.

[6] For fire safety, "IoT-Based Fire Detection System" (2021) presents a GSM and Bluetooth-based alert mechanism embedded with NodeMCU. It employs temperature, smoke, and flame sensors to identify fire incidents and sends updates every 15 seconds to the ThingSpeak cloud. It also comprises an Android app for manual patrol support, thus being adaptable for implementation in schools, offices, and industrial areas. However, its GSM dependence and patrolling by hand decrease its viability for large-scale automated deployment.
adjustments or maintenance required.

[7] In mining settings, "IoT-Enabled Helmet to Protect the Health of Mine Workers" offers a novel solution by incorporating environmental and health monitoring into wearable safety equipment. The helmet monitors heart rate, gas concentration, temperature, and others, issuing alerts with GPS-enabled emergency location tracking. It has a 96–99% accuracy rate across different settings. Long battery life, durability, and user comfort issues are still unresolved, however, which suggests the necessity for improved ergonomic and operational designs.

[8] A fascinating paper called "Recognition of IoT-Based Fire Detection System Fire-Signal Patterns Applying Fuzzy Logic" (2023) brings fuzzy logic to the game to minimize false alarms and discover fires as much as 30 seconds sooner than conventional systems. From actual alarm data for five years, the paper asserts a reduction in false alarms by 80%. Yet its evidence spans only one institution, and there are no performance tests across heterogeneous environments or in various integration configurations.

[9] In their paper "LPG Gas Leakage Detection System Using IoT" (2022), the authors present a system to detect LPG leakage, trigger an exhaust fan, and send alerts to users through SMS. Though appropriate for domestic use and small enterprises, the system lacks robustness with respect to GSM dependency and cloud storage for maintaining historical data. Sensor sensitivity issues also lead to false alarms.

[10] Lastly, "Analysis of Fire Risks and Mitigation Approaches in the Apparel Manufacturing Industry" (2023) examines fire risks in an industrial environment, specifically garment factories. It outlines problems such as unsafe storage and poor exits and recommends mitigation measures in accordance with NFPA standards. Although the analysis is useful, it does not have practical application and is based mostly on expert views without data support.

CHAPTER 3
SYSTEM REQUIREMENTS

Creating an IoT-based safety system that detects dangerous gases, fire and smoke in industrial surroundings is not just a case of wires or coding. It is about designing a reliable, responsible and practical solution, which individuals can rely on - especially in the environment with dangerous work. To work well with our project well and to be effective in real situations, we need both a clear operating plan with a smart combination of hardware and software. What is needed to achieve our system here, a friend of it is breaking:

3.1 Software Requirements:

Each reliable hardware setup is a solid software foundation - and in our industrial security system based on IoT is the software side where intelligence, communication and automation. The project brings together many platforms, languages and open-source libraries so that it can be free of problems, react to real time and easily monitor the remote control.

Here is how our system software ecosystem works in a regular human understanding of language:

Development Platforms and Tools:

Arduino IDE: This is a software platform that I use coding and code to upload the ESP32 board. It is easy to use, but is powerful and perfect for troubleshooting and library management.

Blynk Platform: We use BLYNK to develop a smartphone -based interface, where users can see the sensor value (gas level, temperature, etc.) in real time and receive information. This button also supports widgets and information, making it easier to control the system with a smartphone.

Firebase: A powerful cloud platform developed by Google, Firebase is used to store and sync data (eg fire register or gas level) in real time. It also provides an expandable backend if we ever want project history trends.

C++: C ++ is used for system logic code writing, as it is the main language used in Arduino-based systems. This sensor is responsible for doing everything from reading to sending a notice.

TWILIO API: Twilio must have an account to provide a key to API to send SMS notifications. The service sends the HTTP request at the closing point. It works without the internet

Libraries Used:

To make the improvement method smoother and greater efficient, we've protected numerous libraries that offer ready-to-use functionalities for sensors, communication, and connectivity. Here are the important libraries:

Adafruit Unified Sensor – via Adafruit

DHT Sensor Library – for analyzing temperature and humidity statistics

MQUnifiedSensor – by Miguel, for use with MQ-series gas sensors

ESPSoftwareSerial – by Dirk, utilized for serial communication with GSM or GPS modules

Firebase ESP32 Client – by using Mobizt, utilized for controlling all Firebase integration operations.

HTTPClient Library - A part of the ESP32 Arduino core, employed for the transmission of SMS requests to the Twilio API.

Blynk – by Volodymyr, a central library for BLYNK.

These libraries provide the foundation on which sensor statistics are rendered comprehensible, transportable over the net, and obvious for hardware to cloud/cellular communication.
Together, this software program set makes the system responsive, networked, and smooth to remotely monitor, however light-weight and rapid sufficient to be run on an embedded microcontroller. With the mixing of gas sensing, place-based monitoring, or the gathering of incident data over the internet, this software tool set and lib.

3.2 Hardware Requirements:

Hardware has ears and arms on our device. It assesses the risk, reacts, and makes the people around it privy to the time.

Sensors:

MQ-135 Gas Sensor: It identifies different kinds of risky gases such as ammonia, nitrogen dioxide, benzene, smoke, and carbon dioxide (CO_2). It is vital to come across the dimensions of indoor air excellent and poisonous gases.

Fig 3.2.1 MQ-135 Gas Sensor

MQ-2 Gas Sensor: LPG, methane, alcohol, propane, hydrogen, and smoke detection are known to it, facilitating the detection of fuel leaks and detecting gases.

Fig 3.2.2 MQ-2 Gas Sensor

Flame Sensor: It is used to locate the infrared (IR) mild of flames. When you sense a flame, it triggers the microcontroller to light an alarm.

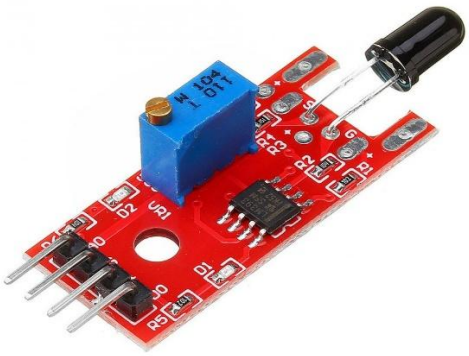

Fig 3.2.3 Flame Sensor

DHT11 Sensor: The sensor provides records on temperature and humidity in real-time, which can be used for fire risk estimates and popular environmental tracking.

Fig 3.2.4 DHT11 Sensor

Alert System:

LED Indicators: Color LEDs are used to visually depict the condition of the system -normal operation, gas leak detection or fire. They are still effective in drawing attention.

Fig 3.2.5 LED Indicator

Buzzer: Sound alarms that give a high alert when some risky situation is detected. This ensures that people who do not participate in the system become careful with potential danger.

Fig 3.2.6 Buzzer

Processing Unit:

ESP32 Dev Module: It is the brain of our Setup-A Strong, Wi-Fi and Bluetooth-supported microcontroller sensor responsible for the treatment of entrance, acts on output and interacts with clouds and mobile platforms.

Fig 3.2.7 ESP32 Dev Module

3.3 Methodology & Dataset:

How our system works, it is okay, effective and time-saving

Real time monitoring:

The sensors are always turned on and usually scan the surroundings for signs and possibility symptoms that include increased levels of fuel, smoke or fire.

Threshold -based reaction:

In each examiner, the trip goes for a safety limit in the code. When the analyzing goes over the brink, the microcontroller reacts proper away — it sends an alert, triggers the buzzer, and flashes the LED.

Emergency Alerts with Location:

When something is going awry, the device does no longer simply trigger a community alert. It sends an SMS alert with GPS coordinates, and consequently keeps distant overseers, rescue groups, or safety government abreast in actual time — if now not on place.

On-Site Awareness:

Rotating buzzers and flashing LED lights warn workers within the region, giving them a danger to break out or observe safety protocols.

Unlike beyond structures that rely upon scheduled inspection or cached facts, ours runs strictly in actual-time. No waits, no dependencies — best ruthless vigilance and instanaction.

CHAPTER 4
PROPOSED MODEL, MODULES DESCRIPTION AND UML DIAGRAMS

4.0 Proposed Model:

Our IoT-Based Industrial Safety System was built with one aim, which is to protect life and prevent devastation from ever happening. In the hazardous industrial work environment where flammable gases, smoke, and fire are likely hazards, a few seconds of early warning can go a long way. This is why we have developed this intelligent real-time system that continuously monitors and then executes action the moment something goes wrong.
It is not an incomplete statement. The microcontroller, ESP32 is here regarded as the main body of the whole system. Different modules are interfaced to the controller; one of them is given to sense danger and rescue humans. The basic components include:
Gas Detector
Fire Detector and Smoke Detector
Alert System and Notification
Controller Unit

All these modules rely on one another's functions and form together as a single safety team. The gas detection module continuously looks for hazardous gases like carbon monoxide (CO), and volatile organic compounds (VOCs) through the MQ-135 sensor. The fire and smoke detection module looks through its vigilant eyes, searching for enviable fire or thick particulates, which are very valuable indicators of an imminent fire.

When there is any deviation from the normal situation, the control unit shall react. It reads sensors and reacts calls an immediate response to turn on a high frequency buzzer illuminates LED's or alarms. Alarms are sent through SMS to the emergency contacts using the Twilio API so that people can be contacted in a hurry. Locally, there are indications of alarms such as a buzzer or LED's that workers can refer to push through.

4.1 Modules Description:

1. Gas Detection Module:

This module is the device's "nose." It makes use of an MQ-a hundred thirty-five fuel sensor to sniff out harmful substances like carbon monoxide, ammonia, and VOCs. It works constantly checking air first-class and sending readings to the microcontroller. When dangerous gasoline stages are detected, it triggers an alert immediately. This helps prevent inhalation risks and capacity explosions due to flammable gases.

2. Fire and Smoke Detection Module:

This is the "watchdog" in the system. These are:

• A flame sensor for detecting infrared radiation from open flames.

• A smoke sensor for detecting air particles indicating the early stages of the fire.

In case of fire or smoking detection, this module works immediately and triggers signals to the control unit - preventing the fire from spreading.

3. Control Unit:

This is the brain in the system. Depending on an ESP32 or Arduino Uno, the control unit constantly receives information from all sensors, compares it with pre-installed secure thresholds, and takes action as needed. If any reading from the sensor is outside the danger label, it ensures that the alarm is active without hesitation. The programming is built into firmware and is performed automatically - no human intervention is necessary when served.

How It All Comes Together:

Each module is designed to perform a task - and it is performed effectively. But how they cooperate that make the system strong. The sensor feeling determines the control unit and contracts the warning system. Whether it is a gas leak, a spark of fire or smoke in the atmosphere, the system ensures that it is not ignored.

The most important thing is that it is designed for real factory workers, real factories and real situations. This is not proof that any lab-mock-up or concept-this is a tool that will actually save people's lives when implemented in the field.

4.2 UML Diagrams:

4.2.1 System Architecture:

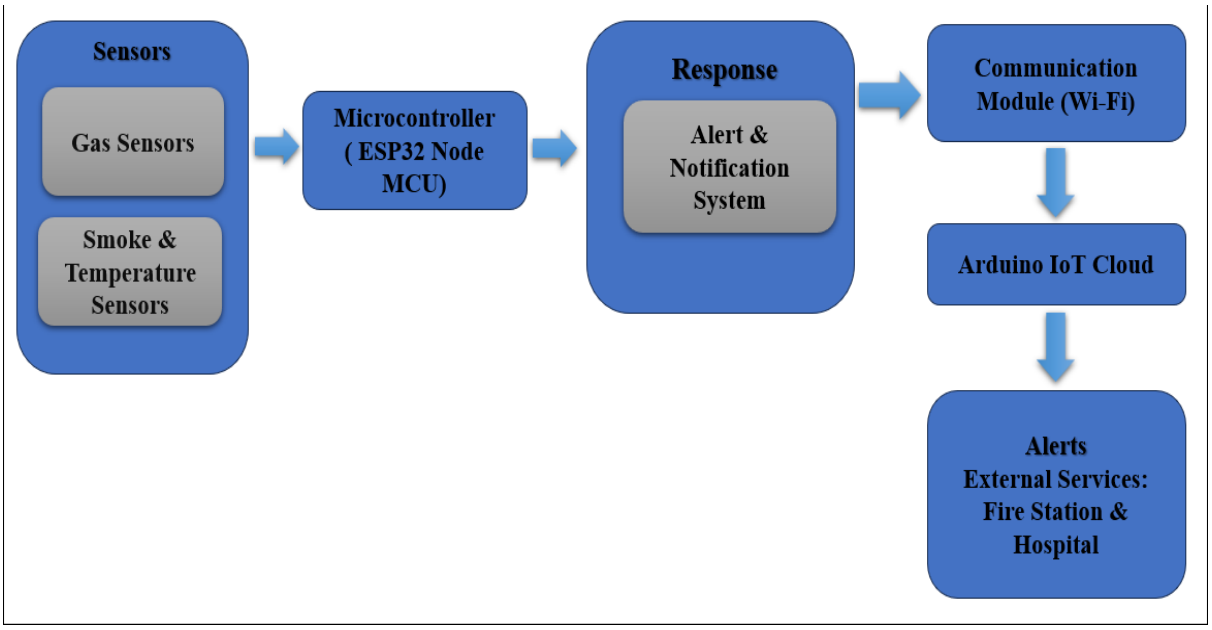

Fig 4.2.1 System Architecture

This system architecture outlines a smart safety monitoring setup designed to detect and respond to potential hazards like gas leaks or smoke. This system is a smart safety setup that uses gas and smoke/temperature sensors to monitor the environment. The data goes to an ESP32 microcontroller, which processes it and triggers alerts if needed. These alerts are sent via a Wi-Fi module to the Arduino IoT Cloud, which then notifies external services like the fire station and hospital for a quick response.

4.2.2 Use Case Diagram:

This use case diagram describes how a safety monitoring system works using sensors for temperature, gas leak, and smoke detection. Anytime there is an event that triggers the sensors, two groups of people are notified:

- Workers and Supervisors: the people onsite, receiving the alerts, that will quickly enter into the site to eliminate the hazard.
- Fire Station and Ambulance: the emergency services that will be contacted off site, and notified to provide professional assistance in emergencies.

This safety monitoring system takes data from the sensors and merges it with the onsite personnel (workers and supervisors) and offsite emergency services (fire station, ambulance) the system provides a more accurate way of doing incident response.

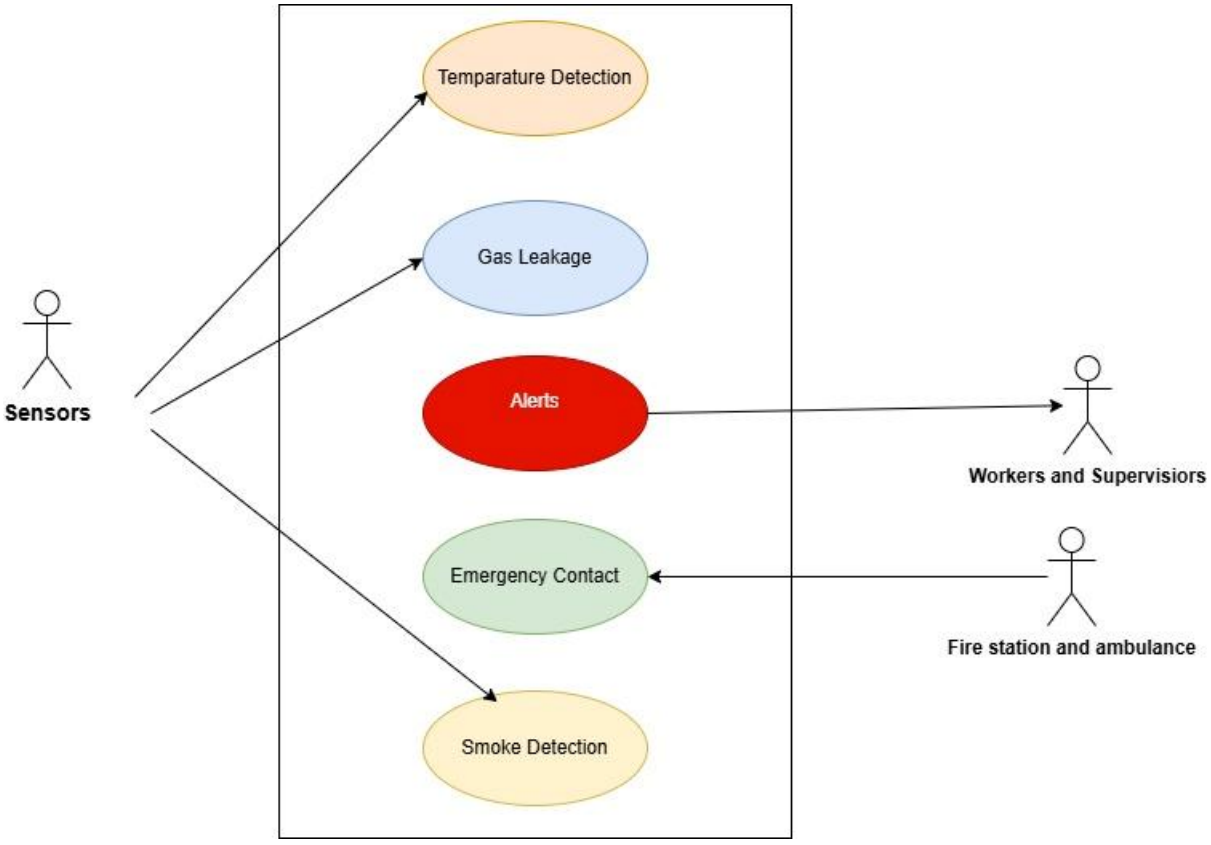

Fig 4.2.2 Use Case diagram

4.2.3 Class Diagram:

This class diagram shows a safety system run by a System Controller that tracks sensors and triggers alerts. The Sensors base class stores data as doubles and uses readData() to gather info. It branches into Temperature Sensor (with readValue()), Flame Sensor (detectFlame()), Gas Sensor (detectGas()), and Smoke Sensor (detectSmoke()), each with thresholds. The Alert System links to the controller, using LED_bool, Buzzer_bool, and sendLocation() to warn about hazards like flames, gas, smoke, or temperature spikes.

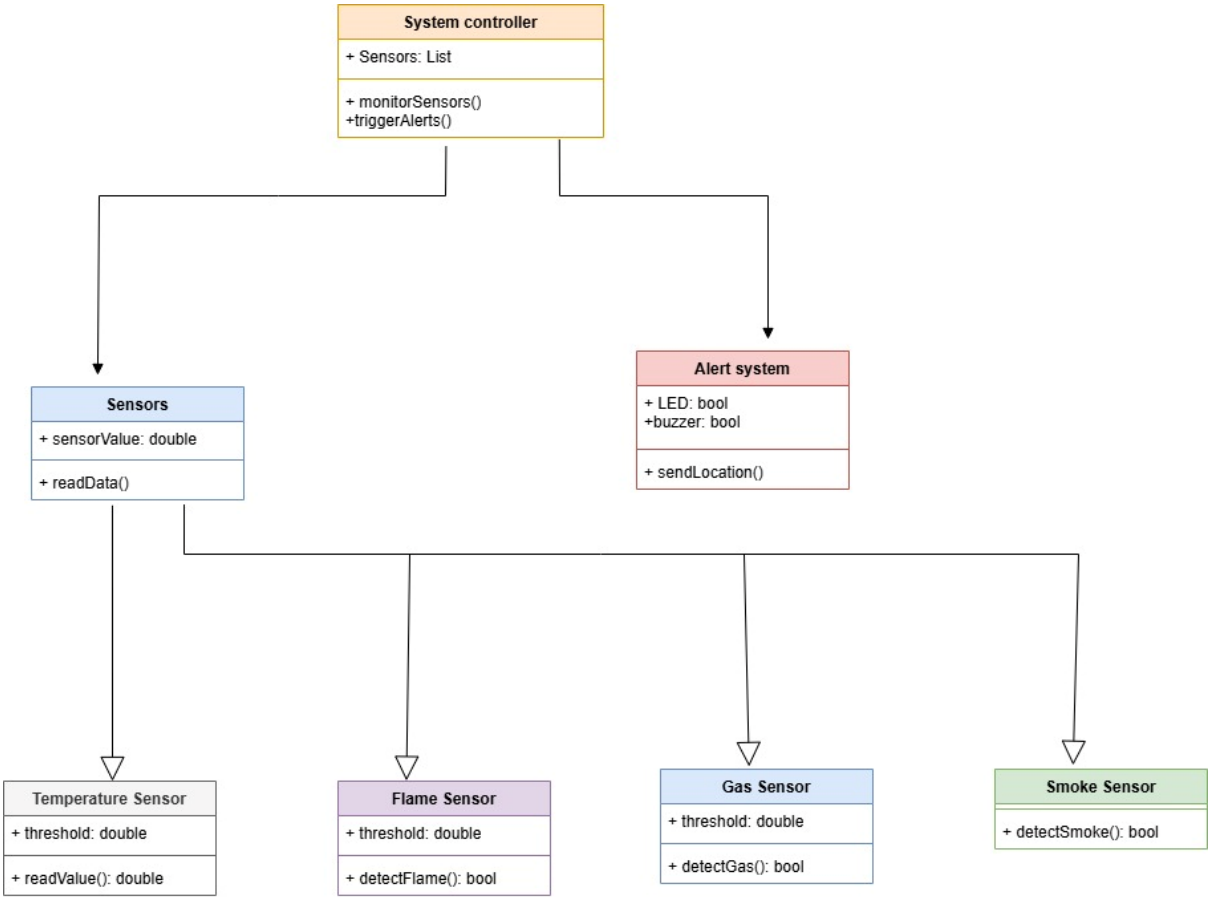

Fig 4.2.3 Class Diagram

4.2.4 Sequence Diagram:

The safety monitoring process presented in this sequence diagram consists of four key actors: Sensors, Controller, AlertSystem, and EmergencyServices. Sensors detect hazards and send readValues() command to the Controller which processes the received data. The first processing step checkIfSafe() checks whether or not the situation is safe and will trigger if hazards are detected. When the Controller detects a hazard, it triggers the AlertSystem which in turn sends an emergency alert to EmergencyServices. Once the situation is resolved the Controller sends the resolveAlert() message to the AlertSystem. This effectively closes the loop. Now hazards can be detected and responded to in a timely manner.

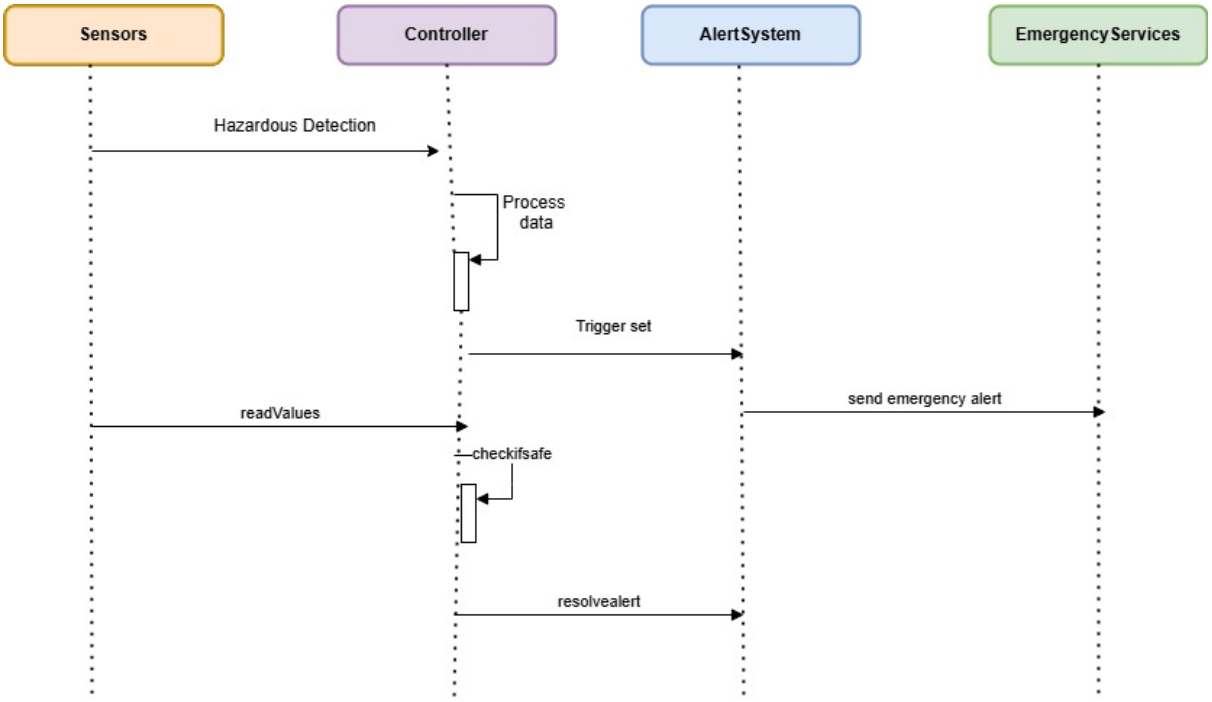

Fig 4.2.4 Sequence Diagram

CHAPTER 5

IMPLEMENTATION, EXPERIMENTAL RESULTS & TEST CASES

CODE:

Mini.ino

```
/**
 * Industrial Safety System v1.0
 * Monitors temperature, humidity, gas, air quality, and flame using ESP32.
 * Sends SMS alerts via Fast2SMS and logs data to Firebase.
 * Displays values on Blynk app.
 *
 * Hardware:
 * - ESP32
 * - DHT11 (GPIO 32)
 * - MQ2 Gas Sensor (GPIO 34)
 * - MQ135 Air Quality Sensor (GPIO 35)
 * - Flame Sensor (GPIO 19)
 * - Buzzer (GPIO 27)
 * - LED_FIRE (GPIO 5)
 * - LED_SMOKE (GPIO 4)
 *
 * Prerequisites:
 * - Install Arduino IDE with ESP32 board support
 * - Install libraries: Blynk, DHT, FirebaseESP32, HTTPClient
 * - Fast2SMS account with ₹100+ wallet recharge (required for SMS)
 * - Blynk app with template ID: TMPL3Loq7Wyou
 * - Firebase project with provided credentials
 *
 * Note: Fast2SMS requires initial funding; error 999 if not recharged.
 */

#define BLYNK_TEMPLATE_ID "TMPL3Loq7Wyou"
#define BLYNK_TEMPLATE_NAME "Industrial Safety System"
#define BLYNK_AUTH_TOKEN "**********"

#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <DHT.h>
#include <FirebaseESP32.h>
#include <HTTPClient.h>

const char* ssid = "**********";         // WiFi SSID
const char* password = "*********";        // WiFi Password

// Firebase credentials
#define API_KEY "**********"
#define DATABASE_URL "**********"
#define USER_EMAIL "**********"              // Update with valid email
#define USER_PASSWORD "********"              // Update with valid password

```

Fig 5.1(a) code

Initializing libraries like WiFi, HTTPClient, and Arduino IoT Cloud for connectivity and setting up credentials for WiFi, Firebase, BLYNK and API for Twilio

```
// Fast2SMS credentials
const char* apiKey = "**************************************************";
const char* toFire = "+**********";      // Fire station number
const char* toHospital = "+**********";  // Hospital number

FirebaseData firebaseData;
FirebaseConfig config;
FirebaseAuth auth;

#define MQ2_PIN 34           // MQ2 Gas Sensor
#define MQ135_PIN 35         // MQ135 Air Quality Sensor
#define FLAME_SENSOR 19      // Flame Sensor
#define DHTPIN 32            // DHT11 Data Pin
#define BUZZER 27            // Buzzer
#define DHTTYPE DHT11
#define LED_FIRE 5           // Fire LED
#define LED_SMOKE 4          // Smoke LED

const float latitude = 17.4577;    // Location latitude
const float longitude = 78.2543;   // Location longitude

// Flags to ensure single SMS per event
bool fireAlertSent = false;
bool gasAlertSent = false;

DHT dht(DHTPIN, DHTTYPE);
BlynkTimer timer;
bool buzzerManualOverride = false;

// Function to send SMS via Fast2SMS
void sendSMS(const char* to, const String& message) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = "https://www.fast2sms.com/dev/bulkV2";
    http.begin(url);
    http.addHeader("authorization", apiKey);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    String postData = "message=" + message + "&route=q&numbers=" + String(to);
    int httpCode = http.POST(postData);
    String payload = http.getString();
    if (httpCode == 200) {
      Serial.println("✅ SMS sent to " + String(to) + " with code: " + String(httpCode) + ", Response: " + payload);
    } else {
      Serial.println("❌ SMS failed to " + String(to) + " with code: " + String(httpCode) + ", Response: " + payload);
      if (httpCode == 400 && payload.indexOf("999") > 0) {
```

Fig 5.1(b) code

Defining the fire station and hospital contact details (e.g., phone numbers), pin configurations for sensors (e.g., MQ2 for gas, flame sensor on GPIO 32) and output devices (LED, buzzer on GPIO 5) and location data with latitude and longitude. Functions to send SMS alerts via Twilio (using WiFi and HTTP requests) when hazards are detected, including connection setup, message construction, and response handling.

```
        Serial.println("Error 999: Fund wallet with ₹100+ on Fast2SMS");
      }
    }
    http.end();
  } else {
    Serial.println("❌ WiFi not connected");
  }
}

BLYNK_WRITE(V8) {
  int value = param.asInt();
  buzzerManualOverride = (value == 1);
  digitalWrite(BUZZER, value);
  Serial.println("Buzzer manually set to: " + String(value));
}

// Sensor data collection and alert logic
void sendSensorData() {
  float temp = dht.readTemperature();
  float humidity = dht.readHumidity();
  int gas_raw = analogRead(MQ2_PIN);
  int air_raw = analogRead(MQ135_PIN);
  int flame_status = digitalRead(FLAME_SENSOR);

  // Handle DHT sensor failure
  if (isnan(temp) || isnan(humidity)) {
    Serial.println("❌ DHT sensor failure, using 0 for temp/humidity");
    temp = 0;
    humidity = 0;
  }

  int mq2_ppm = map(gas_raw, 0, 4095, 0, 1000);  // Gas concentration
  int mq135_ppm = map(air_raw, 0, 4095, 0, 1000); // Air quality

  Serial.println("MQ2 PPM: " + String(mq2_ppm));

  // Send data to Blynk
  Blynk.virtualWrite(V1, mq135_ppm);    // Air quality
  Blynk.virtualWrite(V2, mq2_ppm);      // Gas
  Blynk.virtualWrite(V3, temp);         // Temperature
  Blynk.virtualWrite(V4, humidity);     // Humidity
  Blynk.virtualWrite(V5, flame_status == 0 ? 1 : 0); // Flame status
  Blynk.virtualWrite(V6, mq2_ppm > 500 ? 1 : 0);     // Gas leak status
  Blynk.virtualWrite(V7, latitude);     // Latitude
  Blynk.virtualWrite(V9, longitude);    // Longitude
```

Fig 5.1(c) code

Sensor data collection from a DHT sensor (temperature and humidity), MQ2 (gas concentration), MQ135 (air quality), and a flame sensor. Data transmission to Blynk for monitoring, including temperature, humidity, gas concentration, air quality, flame status, and location (latitude/longitude).

```
  // Firebase updates
  if (Firebase.ready()) {
    Firebase.setFloat(firebaseData, "/SensorData/Temperature", temp);
    Firebase.setFloat(firebaseData, "/SensorData/Humidity", humidity);
    Firebase.setInt(firebaseData, "/SensorData/MQ2_PPM", mq2_ppm);
    Firebase.setInt(firebaseData, "/SensorData/MQ135_PPM", mq135_ppm);
    Firebase.setInt(firebaseData, "/SensorData/FlameStatus", flame_status);
    if (flame_status == 0 && !fireAlertSent) {
      Serial.println("Fire detected, sending SMS");
      Firebase.setString(firebaseData, "/Alerts/Fire", "🔥 Fire Detected!");
      sendSMS(toFire, "🔥 Fire Detected! Location: Lat " + String(latitude, 4) + "°, Lon " + String(longitude, 4) + "°");
      fireAlertSent = true;
    } else if (flame_status != 0) {
      fireAlertSent = false;
    }
    if (mq2_ppm > 500 && !gasAlertSent) {  // Lowered threshold for testing
      Serial.println("Gas leak detected, sending SMS");
      Firebase.setString(firebaseData, "/Alerts/GasLeak", "⚠ Gas Leak Detected!");
      sendSMS(toHospital, "⚠ Gas Leak Detected! Location: Lat " + String(latitude, 4) + "°, Lon " + String(longitude, 4) + "°");
      gasAlertSent = true;
    } else if (mq2_ppm <= 500) {
      gasAlertSent = false;
    }
  }

  if (!buzzerManualOverride) {
    digitalWrite(LED_FIRE, flame_status == 0 ? HIGH : LOW);
    digitalWrite(LED_SMOKE, mq2_ppm > 500 ? HIGH : LOW);
    digitalWrite(BUZZER, (flame_status == 0 || mq2_ppm > 500) ? HIGH : LOW);
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);

  pinMode(FLAME_SENSOR, INPUT);
  pinMode(BUZZER, OUTPUT);
  pinMode(LED_FIRE, OUTPUT);
  pinMode(LED_SMOKE, OUTPUT);
  digitalWrite(BUZZER, LOW);
  digitalWrite(LED_FIRE, LOW);
  digitalWrite(LED_SMOKE, LOW);

```

Fig 5.1(d) code

Updating Firebase with sensor data like temperature, humidity, MQ2 gas concentration, MQ135 air quality, and flame status.

```
  dht.begin();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");

  Blynk.config(BLYNK_AUTH_TOKEN);
  if (Blynk.connect()) {
    Serial.println("Blynk connected");
  } else {
    Serial.println("❌ Blynk connection failed");
  }

  config.api_key = API_KEY;
  config.database_url = DATABASE_URL;
  auth.user.email = USER_EMAIL;
  auth.user.password = USER_PASSWORD;
  if (Firebase.begin(&config, &auth)) {
    Serial.println("Firebase connected");
  } else {
    Serial.println("❌ Firebase connection failed");
  }

  timer.setInterval(10000L, sendSensorData); // 10s interval for real-time monitoring
}

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

Fig 5.1(e) code

Initializing the DHT sensor and establishing WiFi connection, with a delay and confirmation message if connected.

EXPERIMENTAL RESULTS:

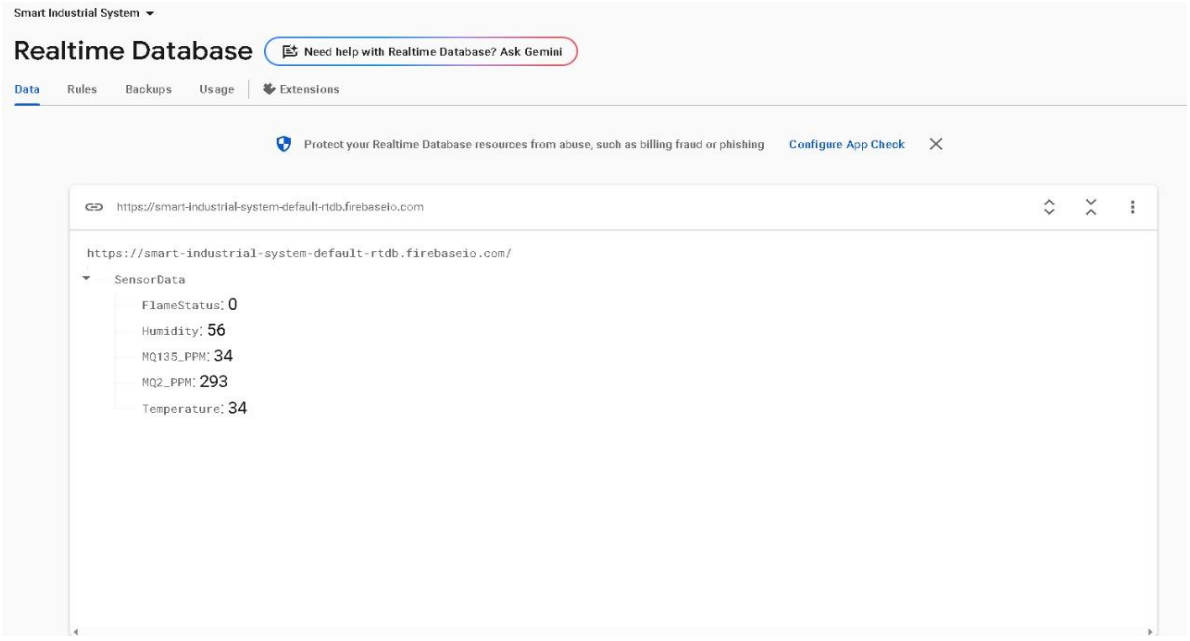

Fig 5.2.1(a) Database values

Firebase Realtime Database interface for the system, showing real-time data.

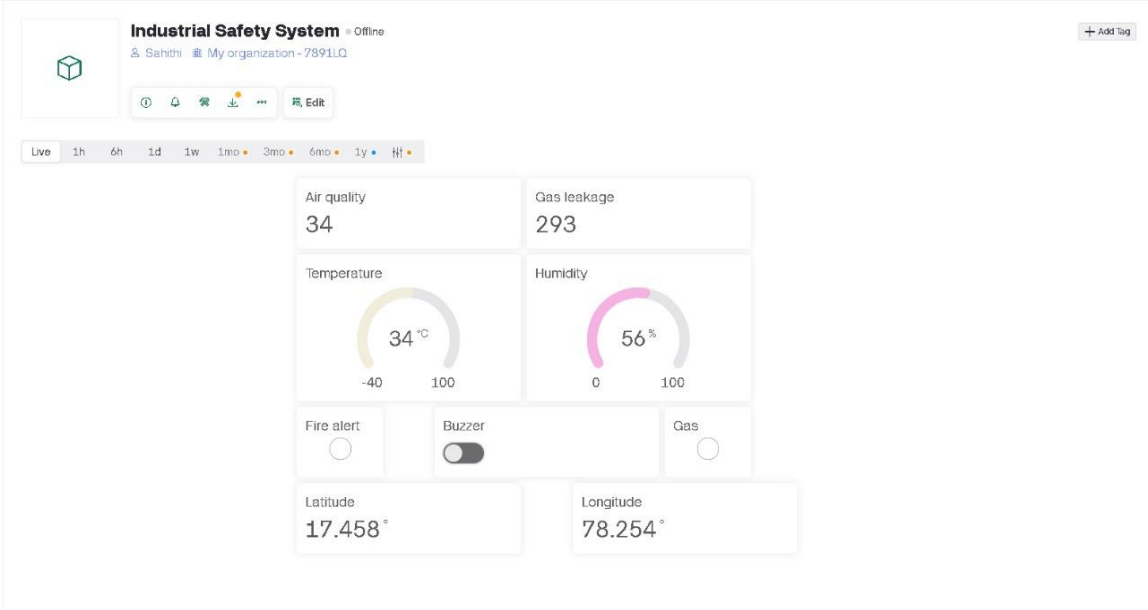

Fig 5.2.1(b) System output

Blynk dashboard for Real-time monitoring.

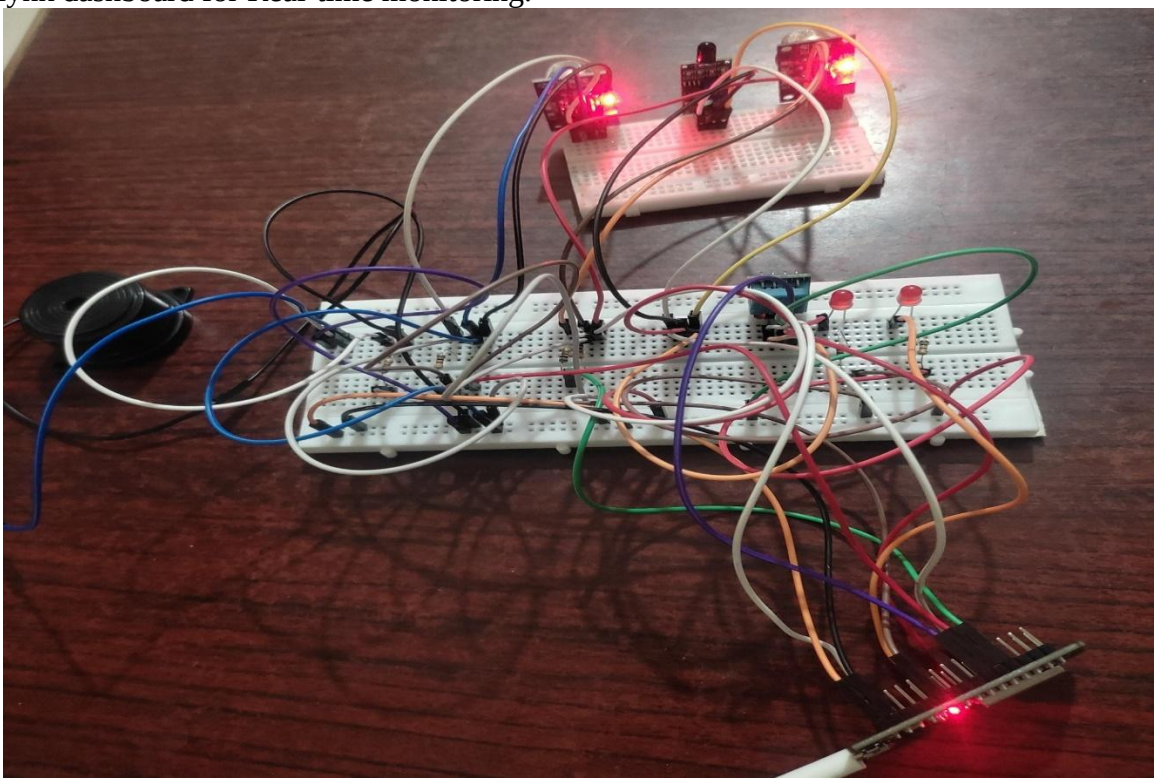

Fig 5.2.1(c) Working prototype (No fire and gas)

The physical setup when no fire and gas is triggered.

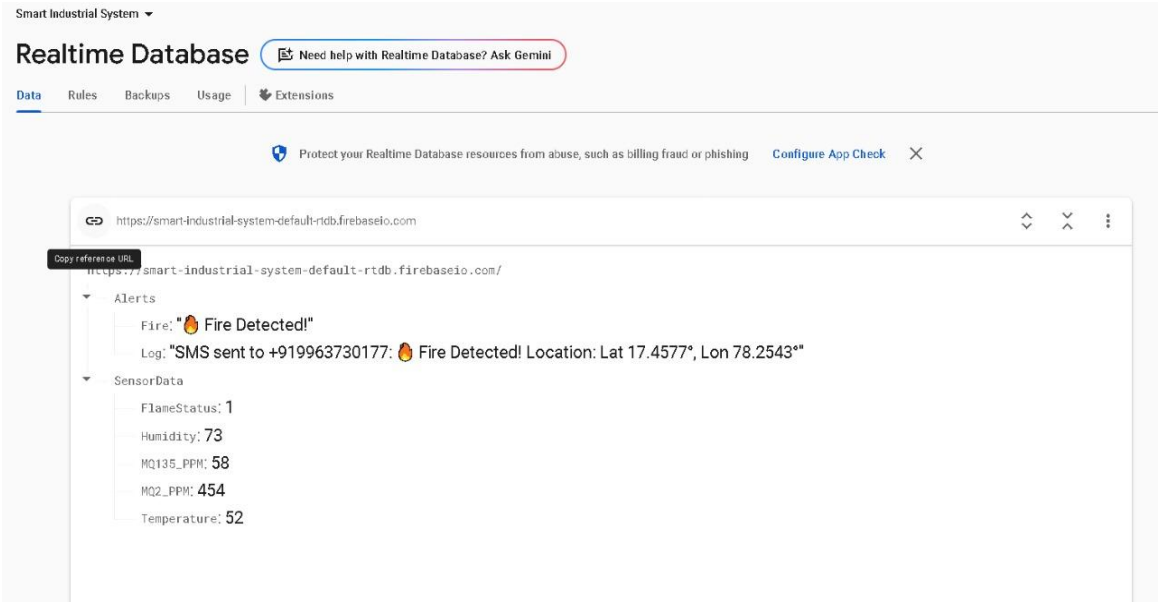

Fig 5.2.2(a) Database values

Firebase Realtime Database interface showing the values when the fire is detected.

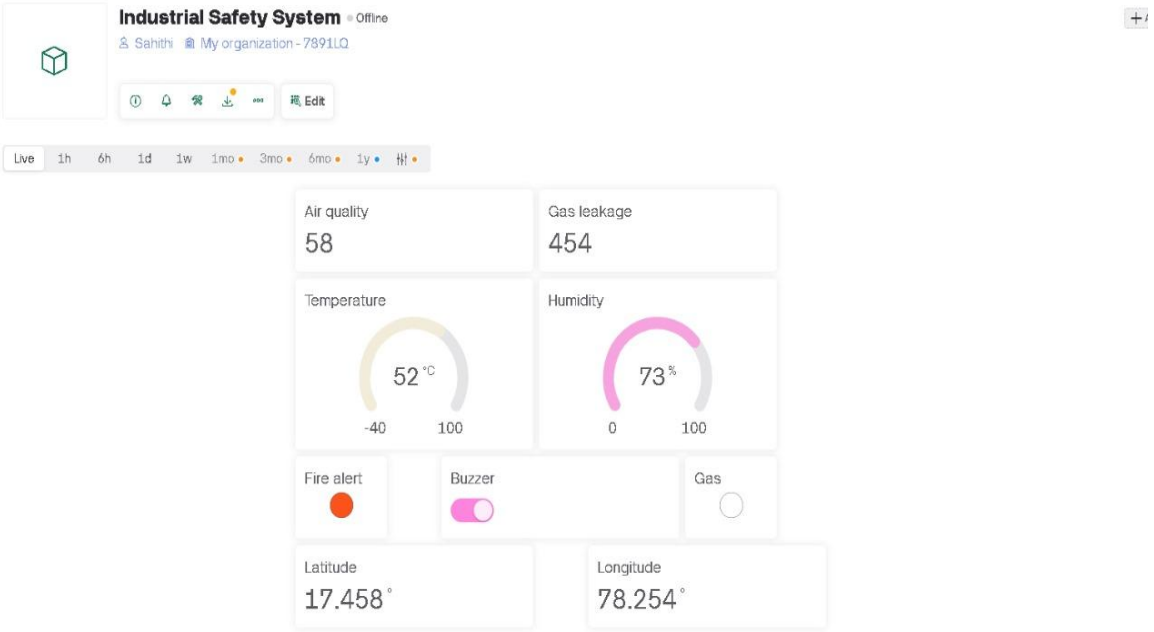

Fig 5.2.2(b) System output

Blynk dashboard showing the fire alert.

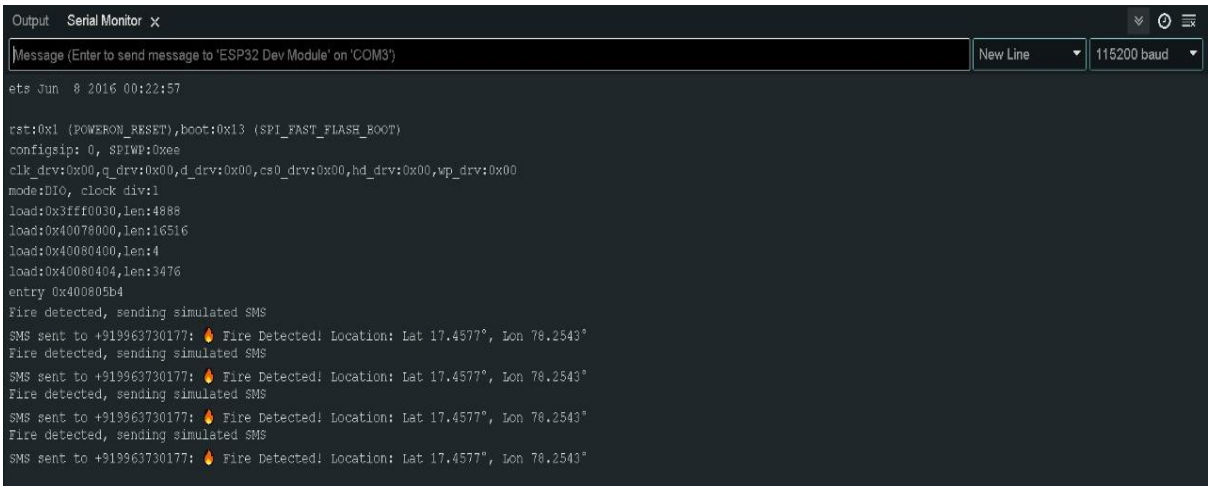

Fig 5.2.2(c) System reaction (Fire and gas detection)

Serial monitor output.

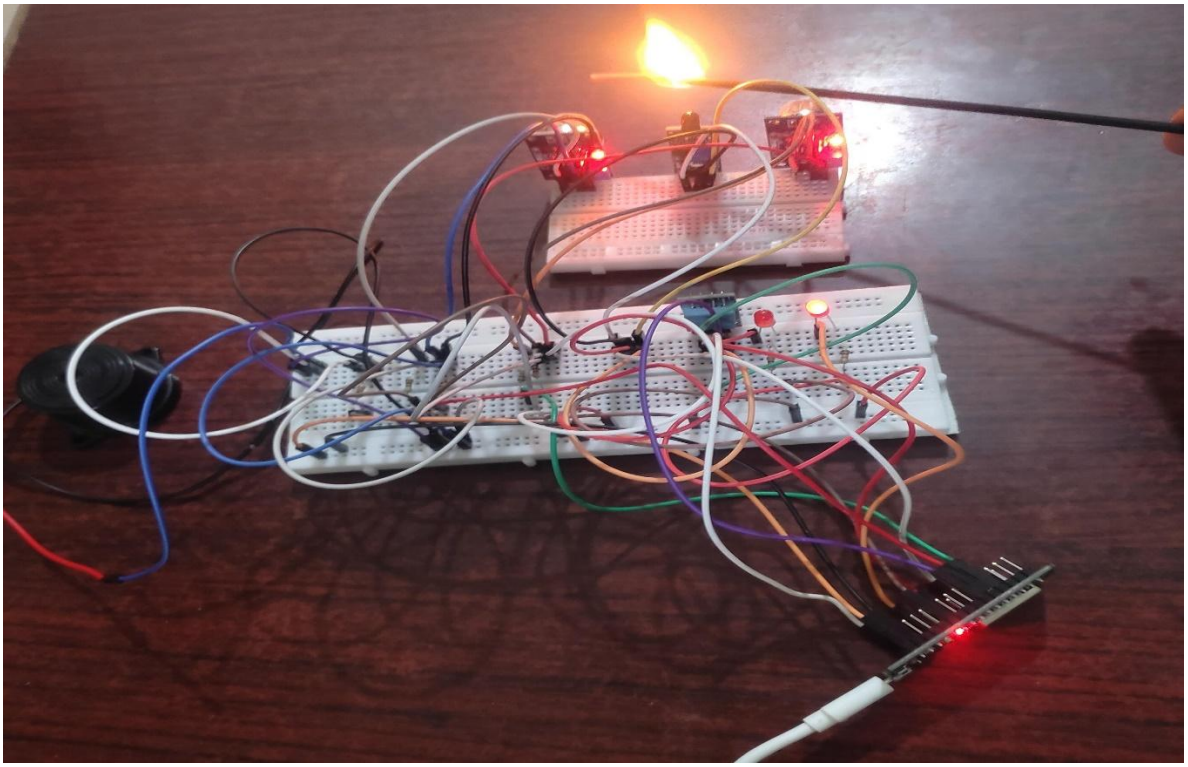

Fig 5.2.2(d) Working prototype (Fire detection)

The physical setup where fire being brought towards the sensor.

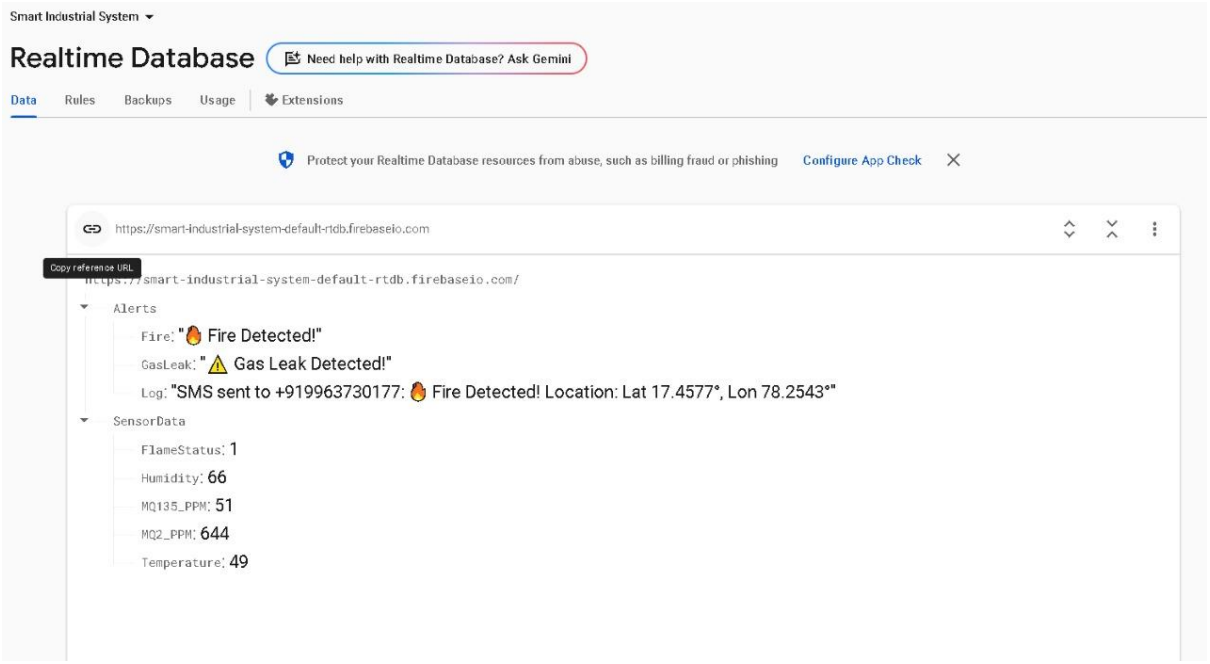

Fig 5.2.3(a) Database values

Firebase Real-time Database interface showing the values when the fire and gas are detected.

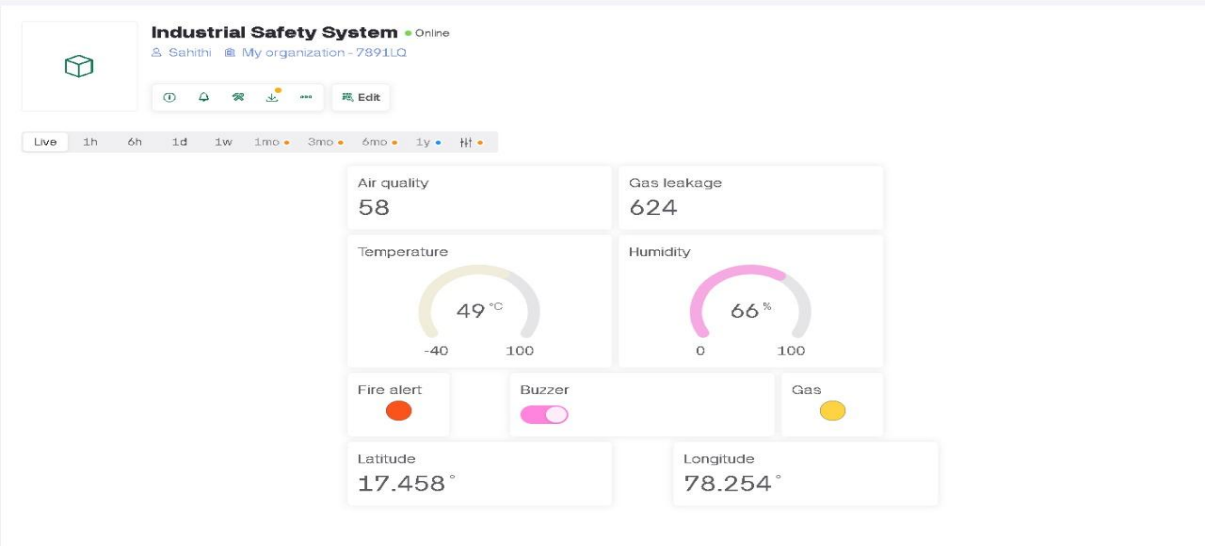

Fig 5.2.3(b) System output

Blynk dashboard for Real-time monitoring.

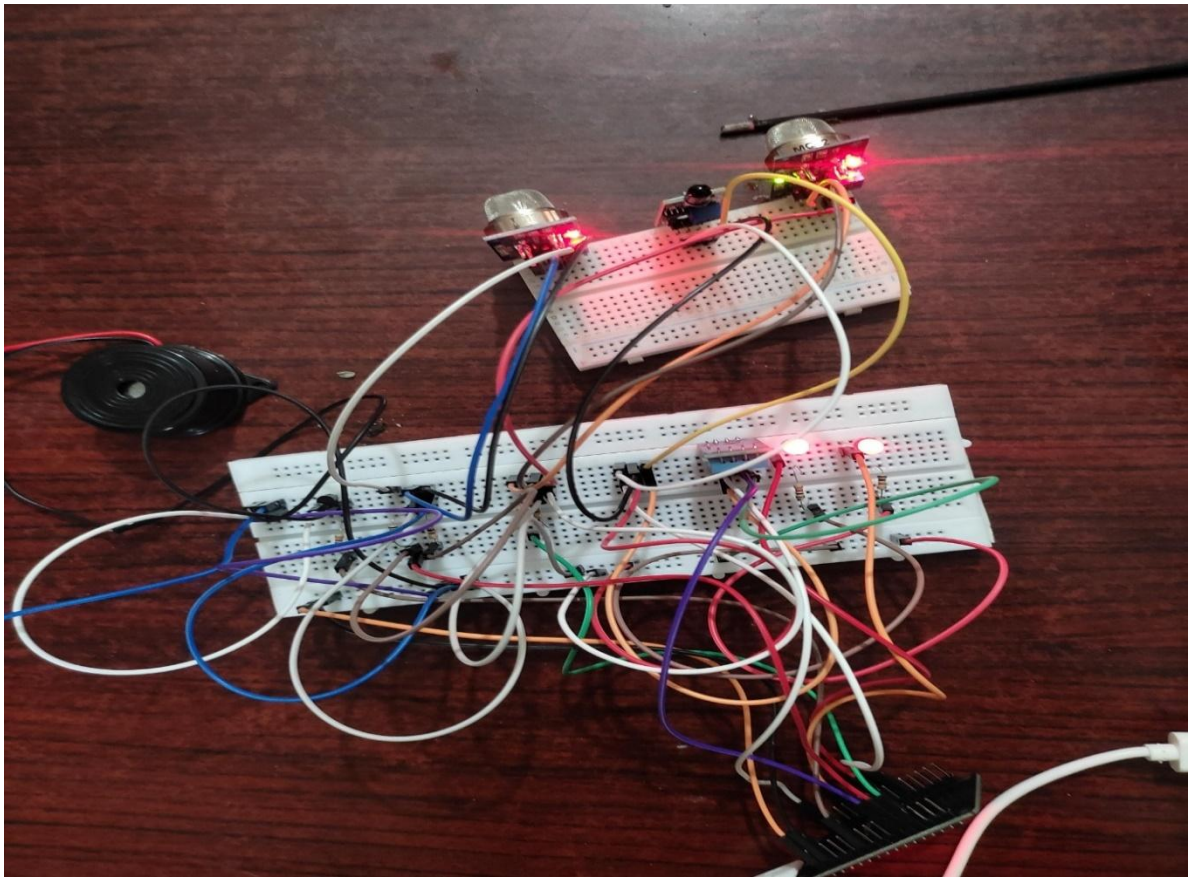

Fig 5.2.3(c) Prototype working

The physical setup where both and fire and gas are triggered.

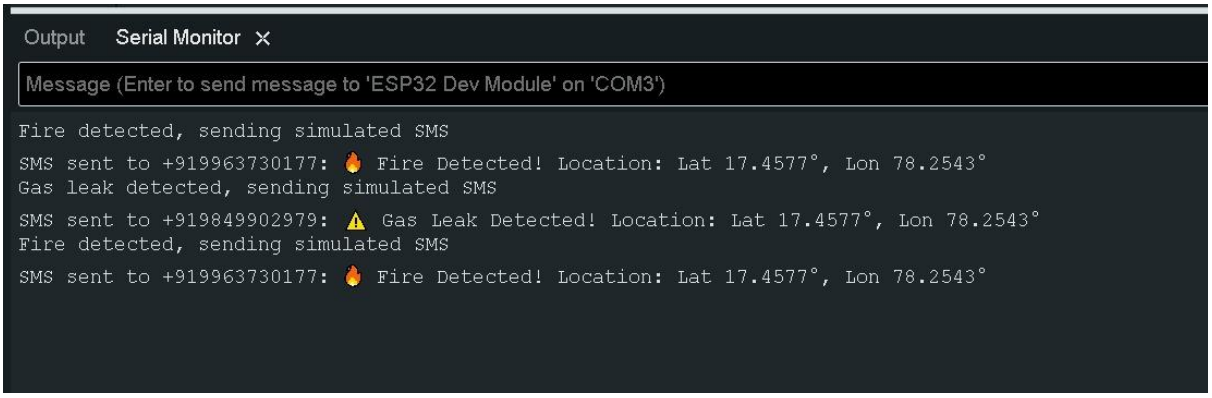

Fig 5.2.3(d) System reaction (Fire and gas detection)

Serial monitor output.

Test Cases:

SNO	EXPECTED INPUT	EXPECTED OUTPUT	ACTUAL INPUT	ACTUAL OUTPUT	TEST RESULT
1.	Room condition (no gas and fire)	No alerts	No gas and fire leak	No alerts	Pass
2.	Fire triggered	Fire alert signal, SMS sent to fire station	Fire near sensor	Fire detected and SMS is sent	Pass
3.	Gas leak	Gas leak alert. SMS sent to hospital	Gas leakage near sensor	Leakage detected, SMS is sent	Pass
4.	High temperature and humidity	Sensor reads values and displays in Blynk app	High humidity	Humidity upadated in Blynk app	Pass
5.	Disconnect WiFi	Error	WiFi is turned off	Error in serial monitor	Pass
6.	Gas warning, but no fire	Just the gas led and alarm turned on	Gas warning, with no flame	Both gas led and alarm are ringing	Pass
7.	Fire warning, and gas is within limits	Only fire led and alarm turned on	Fire warning with no gas leak	Both Fire led and alarm are ringing	Pass

8.	Incorrect WiFi credentials	WiFi not connected msg in the serial monitor	Wrong WiFi credentials	Error in serial monitor	Pass
9.	firebase login is invalid	Firebase not connected	Incorrect login details	Firebase not connected	Pass
10.	Insufficient Twilio balance	Failed to send a SMS error	Not recharged	Error	Fail

Table 5.1: Test cases

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

Conclusion

In places where deadly gas, smoke, and fire are constant threats, the urgency to have an efficient, secure, and intelligent safety method is more important than ever.

Our IoT-driven industry security system shines when it comes to being smart and keeping folks safe on the job.

Combining live monitoring with immediate alarms, this system makes sure factories can get ahead of any problems fast. It doesn't matter what's happening in the background, the system keeps a keen eye on people, property, and operations.

Ease of use and functionality stand out as two of the system's strongest selling points.

You won't have to mess with complex setups or tons of gadgets.

Because it uses easy-to-find microcontrollers and sensors, the setup won't break the bank making it a solid choice for the smaller shops and mid-size businesses.

When you link up with the Twilio API, it sends out immediate text notifications for emergency numbers. This amps up the speed of reactions without messing up the site.

People working on the ground, they get quick visual and sound warnings from Bazers and LED lights, which means they can react on the spot when it's super important

Future Scope

This system will continue to expand and evolve in the future.

In future additions we will likely integrate cloud-based displays that store and display data over time; allowing companies to analyse safety trends changing them from reactive to proactive, can monitor items constantly, and anticipate hazards before they occur.

If we can also enable Wi-Fi communication. It could be dispersed and not reliant on GSM networks in remote areas or remote work environments.

Also, the possibility of incorporating autonomously learning machine learning.

The system will potentially react to hazards like abnormal gas pressure or fire eruption when the materials are still warm before combustibles get too hot.

We could also potentially add a feature that shouts at you, or even connects to engines that extinguishes fires before it starts; and iterate the process allowing applications for broader

use, and provide better routes for dispersion of alerts and who gets the Information.

Overall, our Internet of Things safety system is a flexible, scalable and life-saving system for all workplaces, that is expandable.
In adapting to technology changing, the system will not only become a protection source but also act as a protector.

CHAPTER 7
REFERENCES

1. K. Kumar, A. Verma, and P. Verma, “IoT-HGDS: Internet of Things integrated machine learning based hazardous gases detection system for smart kitchen,” *Internet of Things*, vol. 25, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S2542660524003378
2. A. I. Sunny, A. Zhao, L. Li, and S. K. Sakiliba, “Low-Cost IoT-Based Sensor System: A Case Study on Harsh Environmental Monitoring,” *Procedia Computer Science*, vol. 184, pp. 410–417, 2021.
3. K. B. C. and V. R., “IoT Based Intelligent Industry Monitoring System,” in *Proceedings of the 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2019, pp. 567–571.
4. R. K. Kodali, G. R. N. V., K. P. Nimmanapalli, and Y. K. Y. Borra, “IoT Based Industrial Plant Safety Gas Leakage Detection System,” in *Proceedings of the 2nd International Conference on Inventive Systems and Control (ICISC)*, 2018, pp. 280–284.
5. A. B. Shingate, M. S. Kalkhaire, A. A. Shendage, and Y. N. Shende, “IoT Based Industrial Gas Leakage Detection,” *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, vol. 11, no. 3, pp. 45–49, 2024. [Online]. Available: https://ijsrset.com/index.php/home/article/view/IJSRSET24113112
6. R. V. Patil, S. F. Jadhav, K. S. Kapse, M. B. Thombare, and S. A. Talekar, “IoT Based Fire Detection System,” *International Journal of Emerging Technology and Advanced Engineering*, vol. 11, no. 5, pp. 120–125, 2021.
7. N. Singh, V. K. Gunjan, G. Chaudhary, R. Kaluri, N. Victor, and K. Lakshmanna, “IoT enabled HELMET to safeguard the health of mine workers,” *Computers & Electrical Engineering*, vol. 102, p. 108228, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0140366422002341
8. S. H. Park, D. H. Kim, and S. C. Kim, “Recognition of IoT-based Fire-Detection System Fire-Signal Patterns Applying Fuzzy Logic,” *Sensors*, vol. 23, no. 3, 2023.
9. B. Jena, S. K. Pradhan, R. Jha, S. Goel, and R. Sharma, “LPG Gas Leakage Detection System Using IoT,” *International Journal of Engineering Research & Technology*, vol. 11,

no. 2, pp. 145–149, 2022.

10. M. T. Siraj, B. Debnath, S. B. Payel, A. M. Bari, and A. R. M. T. Islam, “Analysis of the fire risks and mitigation approaches in the apparel manufacturing industry,” *Heliyon*, vol. 9, no. 10, p. e20312, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844023075205

11.https://books.google.co.in/books?id=JPKGBAAAQBAJ&printsec=copyright&redir_esc=y#v=onepage&q&f=falses

Appendix

Software Installation

1. Install Arduino IDE

1. Visit the official Arduino website: https://www.arduino.cc/en/software
2. Download the version suitable for your operating system (Windows, Mac, or Linux).
3. Install and launch the Arduino IDE.

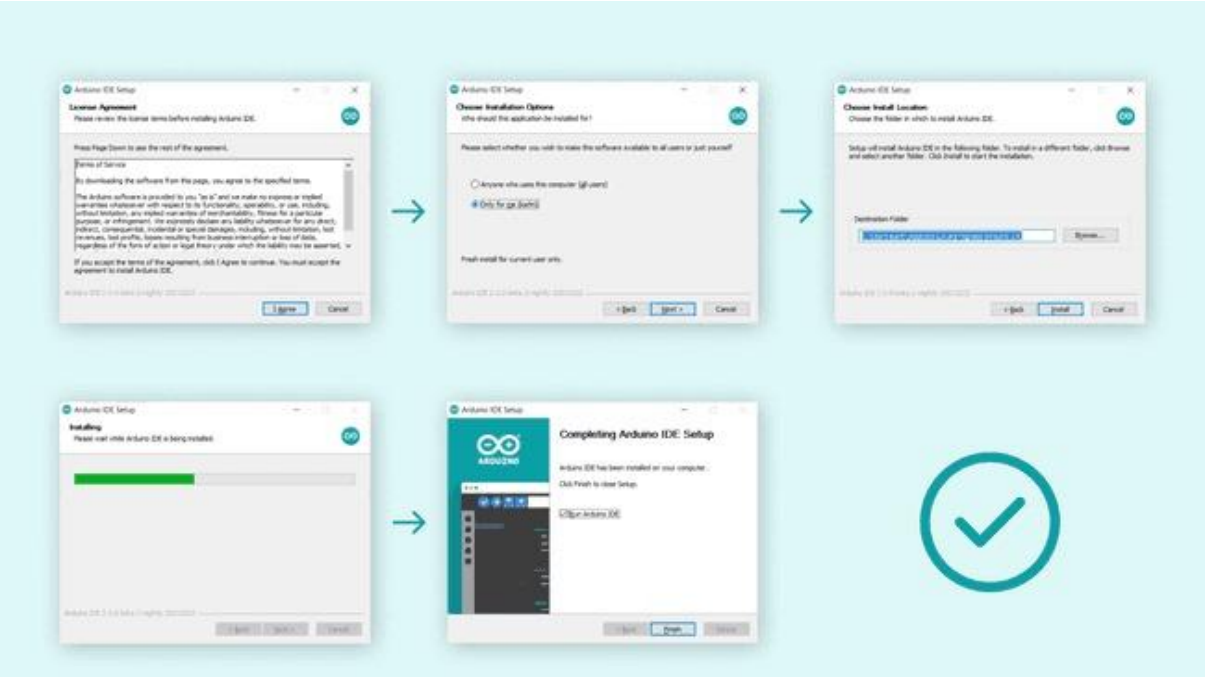

Fig 7.2.1 Arduino IDE Homepage

2. Add ESP32 Board to Arduino IDE

1. Open Arduino IDE.
    a. Go to File > Preferences.
2. In the Additional Board Manager URLs, paste the following URL:https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
3. Go to Tools > Board > Boards Manager, search for ESP32, and install it

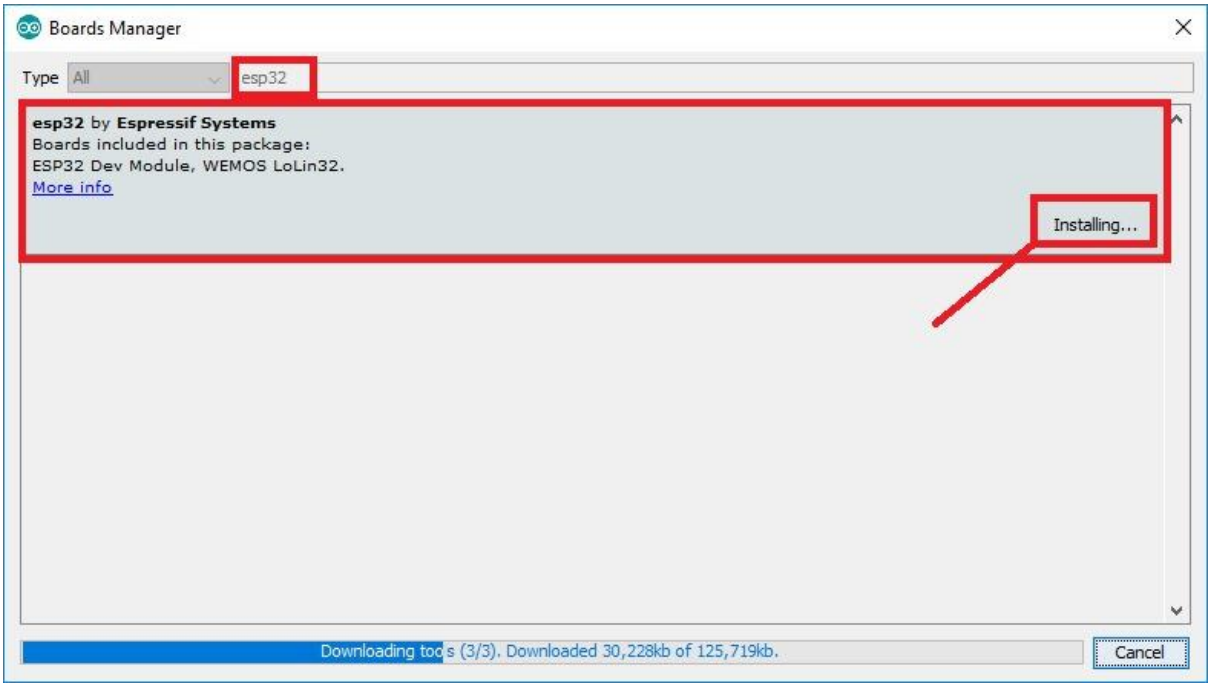

Fig 7.2.2 Preferences Window & ESP32 Installation

3. Install Required Libraries

Use the Library Manager (Sketch > Include Library > Manage Libraries) to install the following:

- Adafruit Unified Sensor
- DHT Sensor Library by Adafruit
- MQUnifiedSensor by Miguel
- TinyGPSPlus by Mikal Hart
- Firebase ESP32 Client by Mobizt
- ESPAsyncWebServer by ESPAsync
- AsyncTCP by dvarrel
- LiquidCrystal I2C by Frank
- LittleFS_ESP32 by lorol
- ESPSoftwareSerial by Dirk
- Blynk Libraries:
    - Blynk by Volodymyr
    - BlynkESP32_BT_WF by Khoi
    - Blynk_Async_ESP32_BT_WF by Khoi
    - BlynkNcpDriver

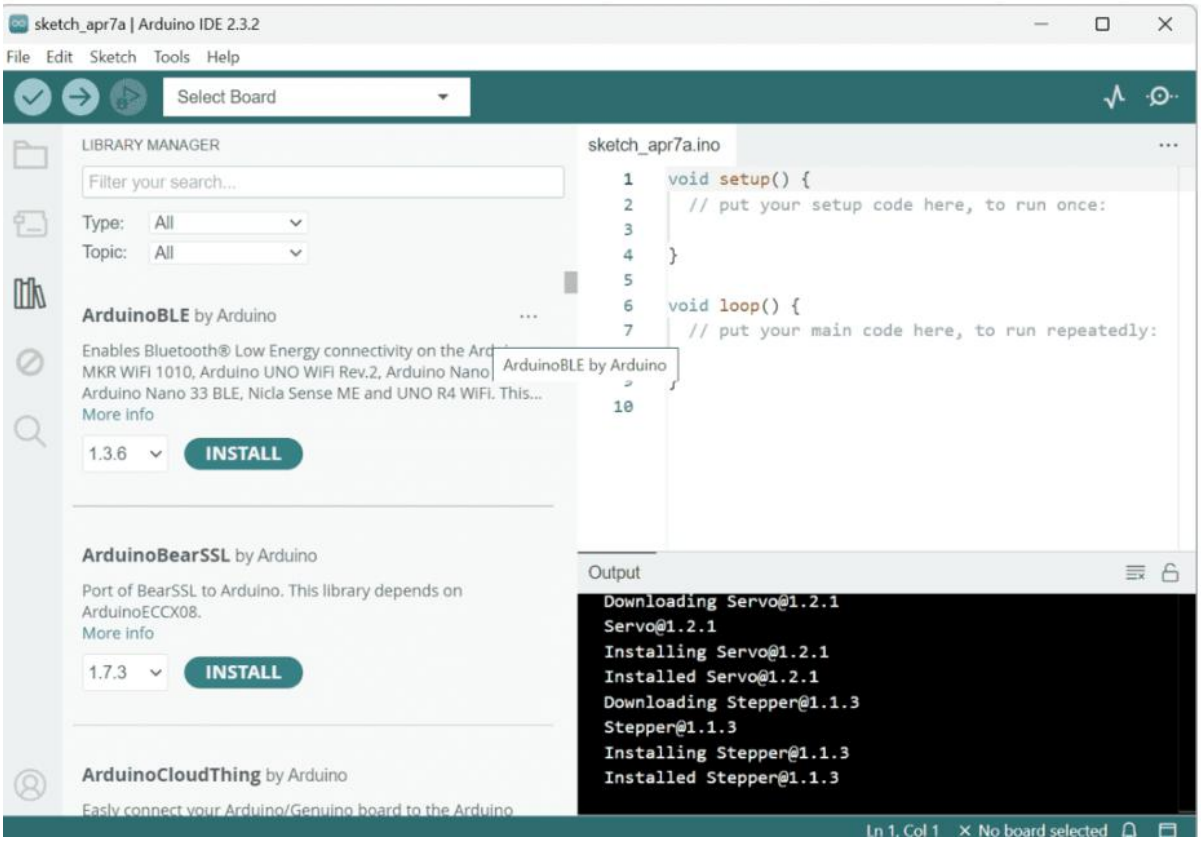

Fig 7.2.3 Library Manager with Libraries

4. Connect ESP32 Board

1. Plug in your ESP32 Dev Module via USB.
2. In Arduino IDE, go to Tools > Board and select ESP32 Dev Module.
3. Select the appropriate COM port from Tools > Port.
4. Open the code pertaining to your project and click on Upload.

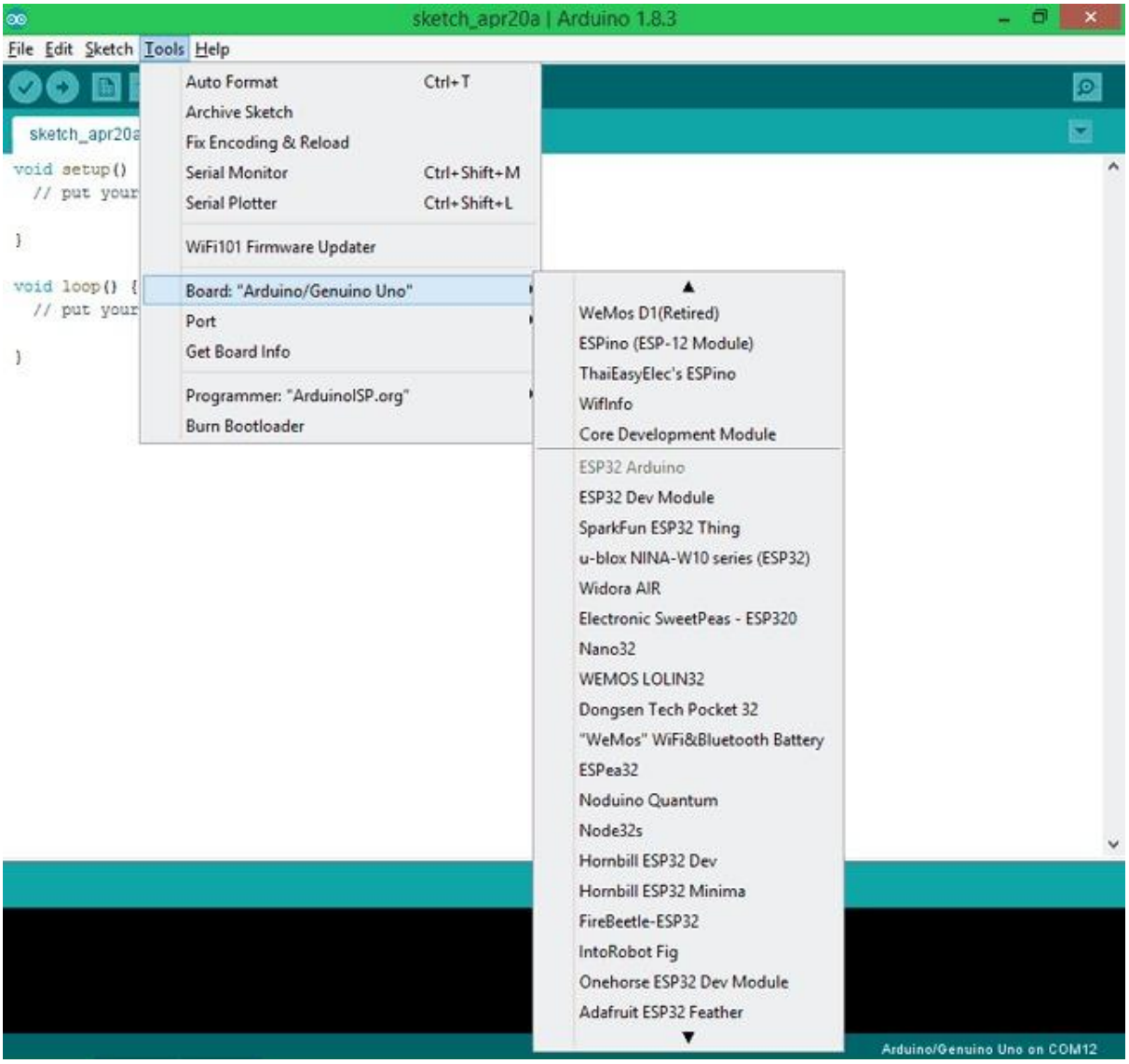

Fig 7.2.4 Connecting ESP32 Boards

5. **Setting up the Blynk Mobile App**
   1. Download Blynk IoT from the Google Play Store or Apple App Store.
   2. For Android: https://play.google.com/store/apps/details?id=cc.blynk
   3. For iOS: https://apps.apple.com/us/app/blynk-iot/id1559314518
   4. Create an account and generate a new template/project.
   5. Add widgets like LCD, buttons, and value displays.
   6. Copy your auth token and paste it in your Arduino code.

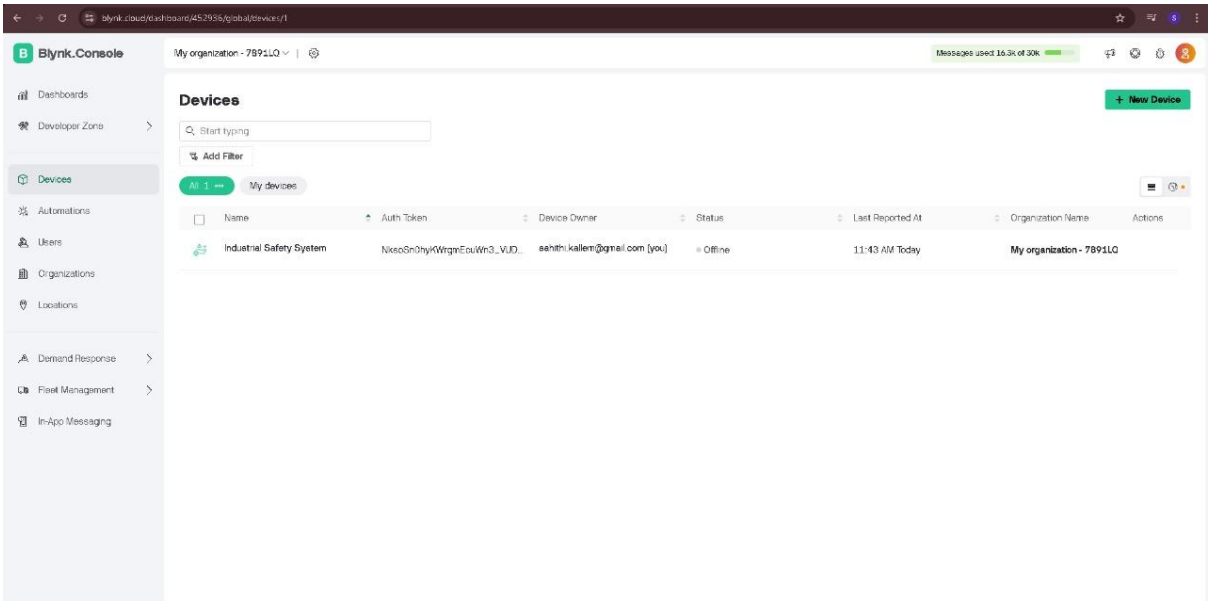

Fig 7.2.5 Blynk Application

6. Setting up Firebase

1. Go to: https://console.firebase.google.com/
2. Create a new project here.
3. Add a real-time database and enable it.
4. Create credentials and update your Firebase token in the code using the Firebase ESP32 Client Library.

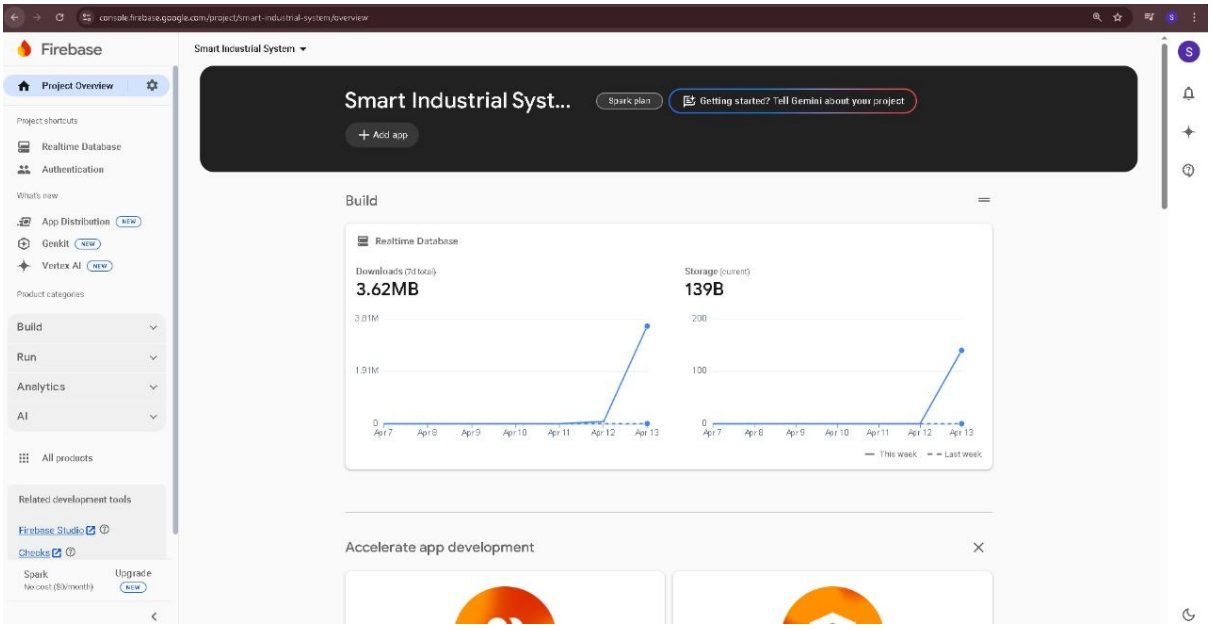

Fig 7.2.6 Firebase Dashboard

7. Final testing

After all installations and settings, supply power to the circuit and test it with the activation of gas, smoke or fire sensors.Check:

Buzzer and LED local alert

SMS alert through GSM module.

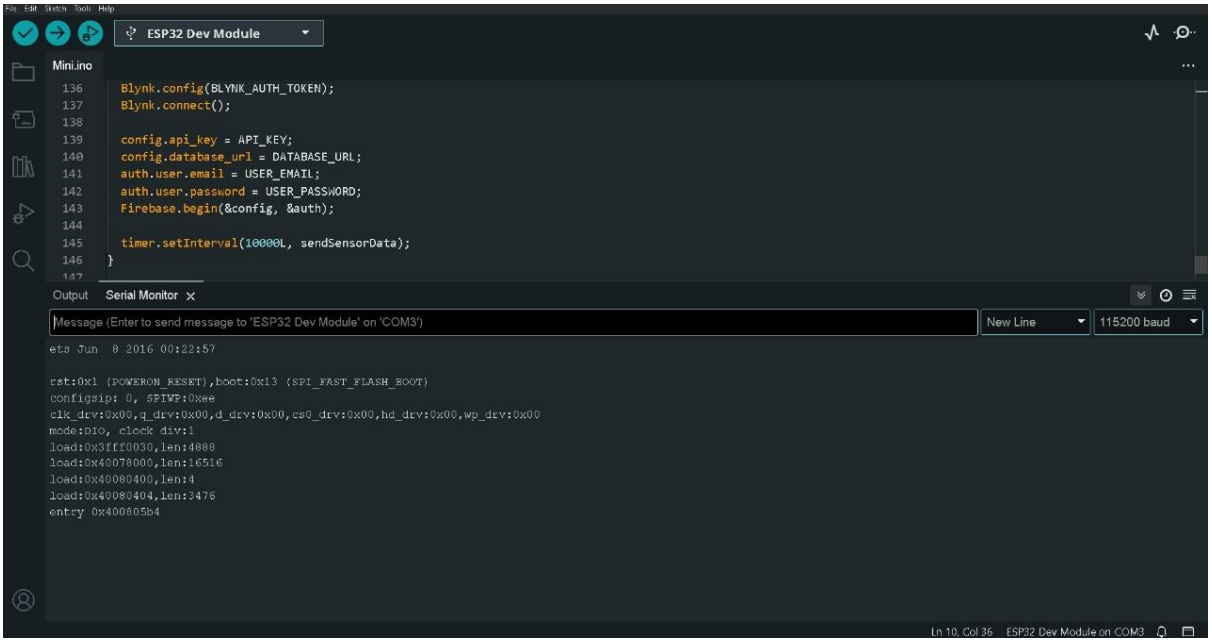

Fig 7.2.7 Serial Monitor

Pseudo code

```
BEGIN

// Initialize WiFi, Blynk, Firebase credentials
SET WiFi SSID and password
SET Blynk Auth Token and Template info
SET Firebase API key, database URL, user credentials
SET Fast2SMS API key and emergency contact numbers

// Define sensor pins
SET MQ2_PIN to 34
SET MQ135_PIN to 35
SET FLAME_SENSOR to 19
SET DHT_PIN to 32
SET BUZZER to 27
SET LED_FIRE to 5
SET LED_SMOKE to 4

// Define static GPS coordinates
SET latitude and longitude

// Flags for alert management
SET fireAlertSent = false
SET gasAlertSent = false
SET buzzerManualOverride = false

// Initialize DHT sensor and Blynk Timer

FUNCTION sendSMS(to, message)
    IF WiFi connected THEN
```

Fig 7.3.1(a) Pseudocode

```
        INITIALIZE HTTP client
        CREATE POST request with Fast2SMS API
        SEND SMS with message and receiver number
        DISPLAY success or error response
    ELSE
        PRINT "WiFi not connected"
    END IF
END FUNCTION

BLYNK_WRITE(V8)
    SET buzzerManualOverride based on button input
    SET BUZZER state accordingly
    PRINT buzzer state

FUNCTION sendSensorData()
    READ temperature and humidity from DHT
    READ gas values from MQ2 and MQ135
    READ flame detection status

    MAP raw sensor values to PPM scale

    SEND sensor values to Blynk
    UPDATE location on Blynk dashboard

    IF Firebase is ready THEN
        STORE all sensor data to Firebase
        IF flame detected AND fire alert not sent THEN
            STORE alert in Firebase
            CALL sendSMS() to fire station
```

Fig 7.3.1(b) Pseudocode

```
            SET fireAlertSent = true
        ELSE IF no flame THEN
            RESET fireAlertSent flag
        END IF

        IF gas PPM > threshold AND gas alert not sent THEN
            STORE alert in Firebase
            CALL sendSMS() to hospital
            SET gasAlertSent = true
        ELSE IF gas PPM <= threshold THEN
            RESET gasAlertSent flag
        END IF
    END IF

    IF NOT buzzerManualOverride THEN
        UPDATE LED_FIRE and LED_SMOKE based on sensor status
        ACTIVATE buzzer if gas or fire detected
    END IF
END FUNCTION

FUNCTION setup()
    INITIALIZE Serial monitor
    SET pin modes for sensors, buzzer, LEDs
    INITIALIZE DHT sensor
    CONNECT to WiFi
    CONNECT to Blynk
    CONFIGURE and CONNECT to Firebase
    SET timer to run sendSensorData() every 10 seconds
END FUNCTION

FUNCTION loop()
    RUN Blynk
    RUN timer
END FUNCTION

END
```

Fig 7.3.1(c) Pseudocode