EXPLOSION RISK DETECTION AND MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

AVINASH N (210701036)

BALAHARINATH C (210701037)

DHARUN PRASATH (210701056)

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE

THANDALAM

ANNA UNIVERSITY

CHENNAI 600 025

MAY 2024

BONAFIDE CERTIFICATE

This is to certify that this project report titled **"EXPLOSION RISK DETECTION AND MANAGEMENT SYSTEM**" is the bonafide work of **"AVINASH N(210701036),BALAHARINATH C(210701037),DHARUN PRASATH(210701056)"** who carried out the project work under my supervision.

SIGNATURE

MR.S.SURESHKUMAR,M.E.,(Ph.D.),

SUPERVISOR
Professor

Department of Computer Science and Engineering

Rajalakshmi Engineering College

Chennai - 602 105

This project report is submitted via viva voce examination to be held on

at Rajalakshmi Engineering College, Thandalam.

EXTERNAL EXAMINER **INTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, I acknowledge the amazing Grace of God Almighty, who blessed my efforts and enabled me to complete this thesis in good health, mind, and spirit.

I am grateful to my Chairman **Mr.S.Meganathan**, Chairperson **Dr.Thangam Meganathan**, Vice Chairman **Mr.M.Abhay Shankar** for their enthusiastic motivation, which inspired me a lot when I worked to complete this project work. I also express our gratitude to our principal **Dr.S.N.Murugesan** who helped us in providing the required facilities in completing the project.

I would like to thank our Head of Department **Dr. P. KUMAR** for his guidance and encouragement for completion of project.

I would like to thank **MR.S.SURESH KUMAR M.E.,(Ph.D.),** our supervisor for constantly guiding us and motivating us throughout the course of the project. We express our gratitude to our parents and friends for extending their full support to us.

AVINASH N(210701036)
BALAHARINATH C(210701037)
DHARUN PRASATH(210701056)

ABSTRACT

The oil and gas industry is inherently prone to explosion risks, primarily due to the potential for leaks. In response to this critical safety concern, we present a comprehensive solution designed to detect and mitigate such risks effectively. Our approach integrates Internet of Things (IoT) technology, machine learning algorithms, and modern web development principles to create a robust and responsive system.

At the core of our solution are sensor-equipped Arduino devices strategically deployed in key areas within oil and gas facilities. These sensors, including UV and MQ2 sensors, continuously monitor for the presence of oil and gas leaks. Upon detecting levels exceeding predefined thresholds, alerts are triggered, initiating swift actions to mitigate potential hazards. The alerts are transmitted via an ESP8266 module, which communicates with a REST API for logging and immediate notification to relevant personnel.To facilitate real-time monitoring and response coordination, we have developed a backend system using the MERN (MongoDB, Express.js, React.js, Node.js) stack. This backend manages the transmission of sensor data, employee information, and system status updates. Leveraging Socket.IO, the communication between the frontend and backend ensures seamless and responsive updates on threat zones and safety protocols. This backend manages the transmission of sensor data.

In response to this critical safety concern, we present a comprehensive solution designed to detect and mitigate such risks effectively. Our approach integrates Internet of Things (IoT) technology, machine learning algorithms, and modern web development principles to create a robust and responsive system. and modern web development principles to create a robust and responsive system.At the core of our solution are sensor-equipped Arduino devices strategically deployed in key areas within oil and gas facilities. These sensors, including UV and MQ2 sensors, continuously monitor for the presence of oil and gas leaks. Leveraging Socket.IO, the communication between the frontend and backend ensures seamless and responsive updates on threat zones and safety protocols. This backend manages the transmission of sensor data.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO.
	ACKNOWLEDGEMENT	**iii**
	ABSTRACT	**iv**
	LIST OF FIGURES	**vii**
	LIST OF TABLES	**viii**
	LIST OF SYMBOLS	**ix**
	LIST OF ABBREVIATIONS	**x**
1	**INTRODUCTION**	**1**
1.1	INTRODUCTION	1
1.2	PROBLEM STATEMENT	2
1.3	SOLUTION	2
1.4	SUMMARY	3
2	**LITERATURE SURVEY**	**5**
2.1	EXISTING SYSTEM	7
2.2	PROPOSED SYSTEM	8
3	**SYSTEM SPECIFICATION**	**9**
3.1	SYSTEM ARCHITECTURE	9
3.2	REQUIREMENT SPECIFICATION	10
3.3	WORKING PRINCIPLE	11

4	**RESULT AND DISCUSSION**	**12**
4.1	ALGORITHM	12
4.2	IMPLEMENTATION	13
5	**OUTPUT**	**14**
5.1	OUTPUTS	14
5.2	SECURITY MODEL	15
6	**CONCLUSION AND FUTURE WORK**	**16**
6.1	CONCLUSION	16
6.2	FUTURE WORK	17
	REFERENCES	**18**
	APPENDIX	**19**

LIST OF FIGURES

FIGURE NO.	NAME OF FIGURES	PAGE NO.
3.1	System Architecture	11
5.1	Output	12

LIST OF TABLE

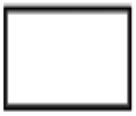
TABLE NO.	NAME OF TABLE	PAGE NO.
4.1	Components Table	14

LIST OF SYMBOLS

Process

This denotes various process involved in the development of proposed system

This arrow indicates the flow from one process to the another process.



This indicates the Stages in the proposed system

ABBREVIATIONS

1. IoT - Internet of Things
2. SDK - Software Development Kit
3. IDE - Integrated Development Environment
4. Wi-Fi - Wireless Fidelity
5. LED - Light Emitting Diode
6. CAD - Computer-Aided Design
7. API - Application Programming Interface
8. USB - Universal Serial Bus
9. GPIO - General Purpose Input/Output
10. MCU - Microcontroller Unit

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Internet of Things (IoT) refers to the network of interconnected devices embedded with sensors, software, and other technologies that enable them to collect and exchange data over the internet. These devices can range from everyday objects like household appliances and wearable gadgets to industrial machinery and smart city infrastructure. The key characteristic of IoT devices is their ability to communicate with each other and with external systems, allowing for remote monitoring, control, and automation of various processes.This connectivity not only enhances convenience for users but also offers opportunities for energy efficiency, security, and personalized experiences tailored to individual preferences.One of the essential components of IoT-enabled devices is the microcontroller, which serves as the brains of the operation, controlling the device's functions and communication with other devices and networks. Arduino UNO is a popular microcontroller board based on the ESP8266 Wi-Fi module, widely used in IoT projects due to its ease of use, built-in Wi-Fi capabilities, and compatibility with Arduino UNO IDE. It features input/output pins for connecting sensors, actuators, and other peripherals, as well as onboard processing capabilities to execute programmed instructions. The key characteristic of IoT devices is their ability to communicate with each other and with external systems, allowing for remote monitoring, control, and automation of various processes. This connectivity not only enhances convenience for users but also offers opportunities for energy efficiency, security, and personalized experiences tailored to individual preferences.

1.2 PROBLEM STATEMENT:

The project addresses the need for improved safety measures in the oil and gas industry by developing an IoT-based explosion risk detection system. Through real-time monitoring and early warning capabilities, the system aims to mitigate potential hazards and enhance the overall safety of industrial operations.

1.3 SOLUTION:

The proposed IoT-based explosion risk detection system for the oil and gas industry is a comprehensive solution designed to significantly enhance safety and operational efficiency. By leveraging advanced IoT technologies, real-time monitoring, predictive analytics, and timely alerts, this system aims to detect potential hazards and prevent catastrophic incidents.

The system starts with the strategic deployment of IoT sensors across various points within the oil and gas facilities. These sensors are tasked with continuously monitoring critical environmental parameters such as gas levels, temperature, and pressure. For instance, MQ2 gas sensors are employed to detect flammable gases, while UV sensors are used to identify the presence of flames. The data collected by these sensors is transmitted to a central processing unit via communication modules like Wi-Fi or LoRa, ensuring seamless and reliable data transfer.

Once the sensor data is collected, it undergoes preprocessing to remove noise and outliers, ensuring accuracy. The data is then analyzed using advanced machine learning algorithms and a rule-based inference mechanism. These tools help in identifying patterns and anomalies that may indicate potential explosion risks. The system applies predefined logic rules and learns from historical data to detect deviations from normal operating

conditions, assessing the severity and proximity of detected risks to critical infrastructure.

In the event of identifying a potential hazard, the system generates real-time alerts. These alerts are communicated to relevant personnel through various channels such as email, SMS, or dedicated monitoring dashboards. The alerting mechanism ensures that appropriate intervention and mitigation measures can be implemented promptly. Additionally, the system features a user-friendly graphical interface for monitoring and managing alerts, configuring settings, and accessing historical data for further analysis and reporting.

Moreover, the system is designed to integrate seamlessly with existing infrastructure and third-party systems within the oil and gas facilities. This interoperability enhances the overall effectiveness of the safety protocols. Regular maintenance and software updates are facilitated through built-in scheduling tools, ensuring the system's continued reliability and optimal performance.

Overall, the proposed IoT-based explosion risk detection system offers a robust solution for enhancing safety measures in the oil and gas industry. By providing real-time monitoring and timely alerts, it significantly reduces the likelihood of incidents, thereby safeguarding personnel and infrastructure..

1.4 SUMMARY:

The IoT-based explosion risk detection system for the oil and gas industry represents a significant advancement in safety and operational efficiency. Leveraging modern IoT technologies, real-time monitoring, predictive analytics, and prompt alert mechanisms, this system is designed to detect and mitigate potential hazards within oil and gas facilities, thereby preventing

catastrophic incidents.

At the core of the system are strategically deployed IoT sensors that monitor critical environmental parameters, such as gas levels, temperature, and pressure. MQ2 gas sensors detect flammable gases, while UV sensors identify the presence of flames. These sensors continuously collect data, which is transmitted to a central processing unit via reliable communication modules like Wi-Fi or LoRa. This constant data flow ensures that any deviation from normal conditions is quickly identified.

Once collected, the sensor data undergoes preprocessing to remove noise and outliers, ensuring accuracy. Advanced machine learning algorithms and a rule-based inference mechanism then analyze the data to identify patterns and anomalies indicative of potential explosion risks. The system applies predefined logic rules and learns from historical data to detect deviations from expected behavior, assessing the severity and proximity of detected risks to critical infrastructure.

In the event of identifying a potential hazard, the system generates real-time alerts. These alerts are communicated to relevant personnel through various channels such as email, SMS, or dedicated monitoring dashboards. This real-time alerting mechanism ensures that appropriate intervention and mitigation measures can be implemented promptly. Additionally, the system features a user-friendly graphical interface for monitoring and managing alerts, configuring settings, and accessing historical data for further analysis and reporting.

CHAPTER 2

LITERATURE SURVEY

1. **Paper:** Design and development of Arduino UNO-based automation home system using the Internet of Things

Author: SA Ajagbe, OA Adeaga, OO Alabi

Year: 2024

Disadvantage: While the system is described as low-cost, it may lack advanced features found in more expensive solutions.

2. **Paper:** An intelligent oil and gas well monitoring system based on the Internet of Things

Author: MY Aalsalem, WZ Khan, W Gharibi

Year: 2017

Disadvantage: The system may face challenges related to data security and the integration of disparate sensor types.

3. **Paper:** A reliable Internet of Things-based architecture for the oil and gas industry

Author: WZ Khan, MY Aalsalem, MK Khan

Year: 2017

Disadvantage: The initial setup and deployment can be costly and require significant technical expertise.

4. **Paper:** Wireless Sensor Networks in the oil and gas industry: Recent advances, taxonomy, requirements, and open challenges

Author: MY Aalsalem, WZ Khan, W Gharibi, MK Khan

Year: 2018

Disadvantage: Managing the energy consumption of sensor nodes remains a significant challenge.

5. **Paper:** A systematic review of big data analytics for the oil and gas industry 4.0

Author: T Nguyen, RG Gosine, P Warrian

Year: 2020

Disadvantage: The implementation of big data solutions can be complex and resource-intensive.

6. **Paper:** The Role of IoT in Optimizing Operations in the Oil and Gas Sector: A Review

Author: SK Sharma, A Rani, H Bakhariya, R Kumar

Year: 2024

Disadvantage: The adoption of IoT technologies may require significant changes to existing infrastructure and processes.

7. **Paper:** Gas Leakage Detection System using IoT with integrated notifications using Pushbullet - A Review

Author: MA Subramanian, N Selvam

Year: 2020

Disadvantage: Reliance on internet connectivity may limit system effectiveness in remote areas.

8. **Paper:** Enabling emergent configurations in the industrial Internet of Things for oil and gas explorations: A survey

Author: OE Ijiga, R Malekian, UAK Chude-Okonkwo

Year: 2020

Disadvantage: Scalability and security issues in large-scale implementations.

9. **Paper:** Oil and Gas 4.0 era: A systematic review and outlook

Author: H Lu, L Guo, M Azimi, K Huang

Year: 2019

Disadvantage: High initial investment for digital transformation.

10. **Paper:** A survey on industry 4.0 for the oil and gas industry: Upstream sector

Author: O Elijah, PA Ling, SKA Rahim, TK Geok, A Arsad

Year: 2021

Disadvantage: Complexity in integrating with existing legacy systems.

2.1 EXISTING SYSTEM:

The current explosion risk detection systems in the oil and gas industry primarily rely on traditional methods and technologies, such as manual inspections and basic sensor networks. These systems typically include gas detectors, flame detectors, and pressure sensors strategically placed throughout the facility to monitor critical parameters. When a sensor detects an abnormal condition, it triggers an alarm that notifies personnel of the potential hazard. While these systems have been effective to some extent, they present several limitations that hinder their overall efficiency and reliability.One of the main drawbacks of existing systems is the lack of real-time data processing and remote monitoring capabilities. Many traditional systems require on-site personnel to respond to alarms, which can delay response times and increase the risk of accidents. Additionally, these systems often generate numerous false alarms due to their inability to differentiate between minor fluctuations and actual hazards, leading to unnecessary downtime and operational disruptions.Another significant limitation is the lack of predictive analytics. Existing systems generally do not utilize historical data to forecast potential risks, making it difficult to implement proactive safety measures. This reactive approach means that potential hazards are only addressed after they have been detected, which can result in significant damage and loss if the response is not swift and effective.Furthermore, the integration of traditional systems with modern infrastructure is often challenging and expensive. Many facilities operate with outdated technology that is incompatible with newer, more advanced safety systems. This incompatibility requires costly upgrades and can create additional operational challenges during the transition period.Overall, while traditional explosion risk detection systems have been foundational in maintaining safety in the oil and gas industry, their limitations highlight the need for more advanced, real-time, and predictive solutions. The integration of IoT technologies, real-time monitoring, and machine learning algorithms offers a promising path forward to enhance safety, reduce response times, and improve operational efficiency.

2.2 PROPOSED SYSTEM:

The proposed system for explosion risk detection in the oil and gas industry is an advanced IoT-based solution designed to enhance safety and operational efficiency. The system integrates a network of IoT sensors, including MQ2 gas sensors and UV sensors, strategically placed throughout oil and gas facilities to continuously monitor critical environmental parameters such as gas concentrations, temperature, and pressure.These sensors are connected to an Arduino Uno R3 and an ESP8266 module, which facilitate data collection and transmission to a central monitoring system via HTTP requests. The central system is built on a MERN (MongoDB, Express.js, React, Node.js) stack, providing a robust backend for data processing and a dynamic frontend for real-time monitoring.The data collected from the sensors undergoes preprocessing to remove noise and ensure accuracy. Advanced machine learning algorithms, specifically linear regression models, analyze the data to predict potential explosion risks based on historical patterns and current readings. This predictive capability allows for early detection of anomalies, providing crucial lead time for preventive measures.Real-time alerts are generated whenever sensor readings exceed predefined safety thresholds. These alerts are communicated to personnel through various channels, including SMS, email, and a dedicated monitoring dashboard powered by Socket.IO for real-time communication. This ensures immediate awareness and response to potential hazards.The user interface is designed to be intuitive, offering comprehensive visualizations of sensor data, system status, and historical trends. This interface allows operators to monitor safety conditions, manage alerts, and access detailed reports, facilitating informed decision-making.Overall, the proposed system aims to significantly reduce the risk of explosions in oil and gas facilities by providing continuous monitoring, predictive analytics, and real-time alerts. This integrated approach not only enhances safety but also improves operational efficiency by enabling proactive maintenance and quick response to emerging threats.

CHAPTER 3

SYSTEM ARCHITECTURE

3.1 SYSTEM ARCHITECTURE

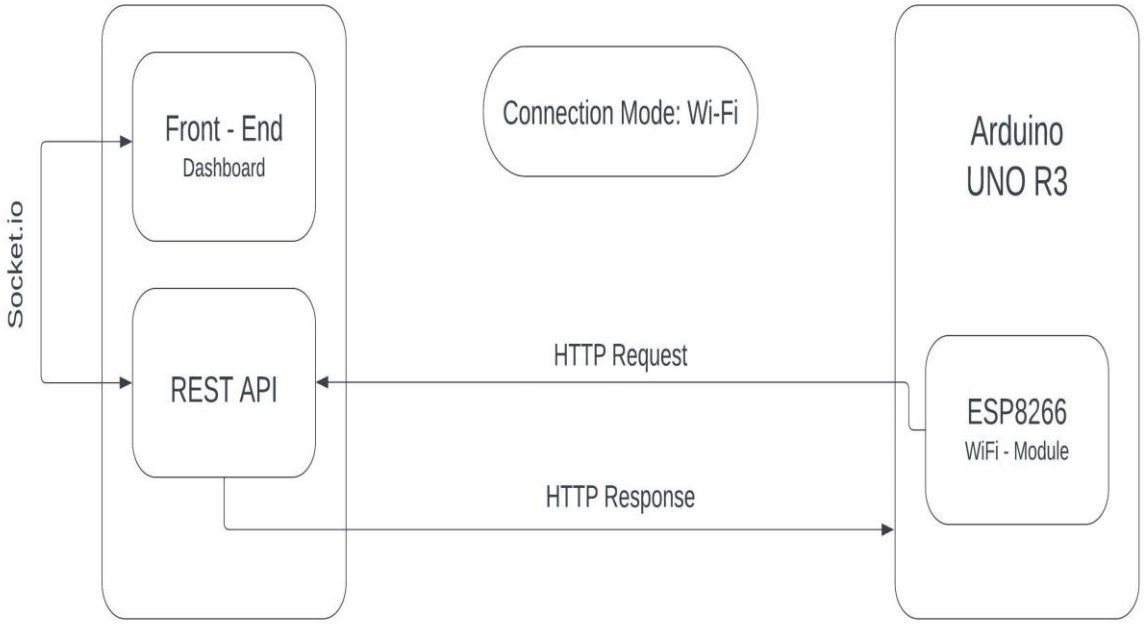


Fig 3.1 System Architecture

3.2 REQUIREMENT SPECIFICATION

3.2.1 HARDWARE SPECIFICATION

Arduino uno r3

Nodemcu v3 esp8266

Mq-2 sensor

Hc-05 sensor

Breadboards and jumpers

16x2 lcd display

Potentiometer

Resistors batteries

3.2.2 SOFTWARE SPECIFICATION

- 8 GB RAM
- Intel core i5
- Node.js with Express.js for backend development
- MongoDB for database management
- Arduino IDE
- React.js for building user interface

3.3 WORKING PRINCIPLE

The working principle of the IoT-based explosion risk detection system in the oil and gas industry revolves around continuous monitoring, data analysis, and real-time alerting to prevent hazardous incidents. This system employs a network of IoT sensors, such as MQ2 gas sensors and UV sensors, which are strategically placed throughout the facility to detect critical environmental parameters, including gas concentrations, temperature, and potential flame presence.These sensors are connected to an Arduino Uno R3, which serves as the primary data acquisition unit. The Arduino processes the raw sensor data and, through the ESP8266 Wi-Fi module, transmits this data to a central server via HTTP requests. This central server is part of a MERN (MongoDB, Express.js, React, Node.js) stack, which handles data storage, processing, and user interface functionalities.Upon reaching the server, the sensor data undergoes preprocessing to filter out noise and correct any anomalies, ensuring high data accuracy. Advanced machine learning algorithms, particularly linear regression models, are then applied to analyze the data. These algorithms predict potential explosion risks by identifying patterns and trends that indicate dangerous conditions based on historical data and real-time readings.When the system detects readings that exceed predefined safety thresholds, it triggers an alert mechanism. Real-time alerts are sent to facility personnel via multiple channels, including SMS, email, and a real-time monitoring dashboard developed using Socket.IO. This immediate notification system ensures that the relevant personnel are promptly informed of any potential hazards, allowing for quick and effective intervention.The user interface of the system is designed to be intuitive, providing operators with real-time data visualization, system status updates, and historical data analysis.

CHAPTER4

RESULT AND DISCUSSION

4.1 ALGORITHM

The algorithm orchestrating the explosion risk detection system in the oil and gas industry follows a meticulous process. It commences with the initialization of sensors (MQ2, UV) and communication modules (Arduino Uno R3, ESP8266), pivotal for continuous data acquisition. Sensor data, encompassing gas concentration, temperature, and UV levels, is transmitted to the central server through ESP8266, where preprocessing is conducted to refine data quality. Leveraging machine learning algorithms, notably linear regression, the system scrutinizes historical and real-time data to pinpoint potential explosion risks. Upon identifying deviations beyond predefined safety thresholds, the system triggers alerts via SMS, email, and a monitoring dashboard. This dashboard serves as a central interface for operators, offering real-time data visualization and historical trends, empowering prompt decision-making and proactive risk mitigation strategies to uphold the safety and integrity of the oil and gas facility..

Component	Function
Arduino UNO	Acts as the microcontroller for system control and facilitates communication with the mobile application.
Mobile Application	Serves as the user interface, allowing remote control of connected devices.

Sensors	Monitor environmental conditions
Power Supply	Provides electrical power to the Arduino UNO and connected devices.
Internet Connectivity	Enables seamless communication between the mobile application and Arduino UNO, allowing remote device control.
Breadboard	Facilitates prototyping and assembling of components.
Jumper Cables	Used for connecting components on the breadboard, aiding in circuit assembly.

Table 4.1Component Table

4.2 IMPLEMENTATION:

The implementation of the explosion risk detection system in the oil and gas industry involves several key steps to ensure its effectiveness and reliability.

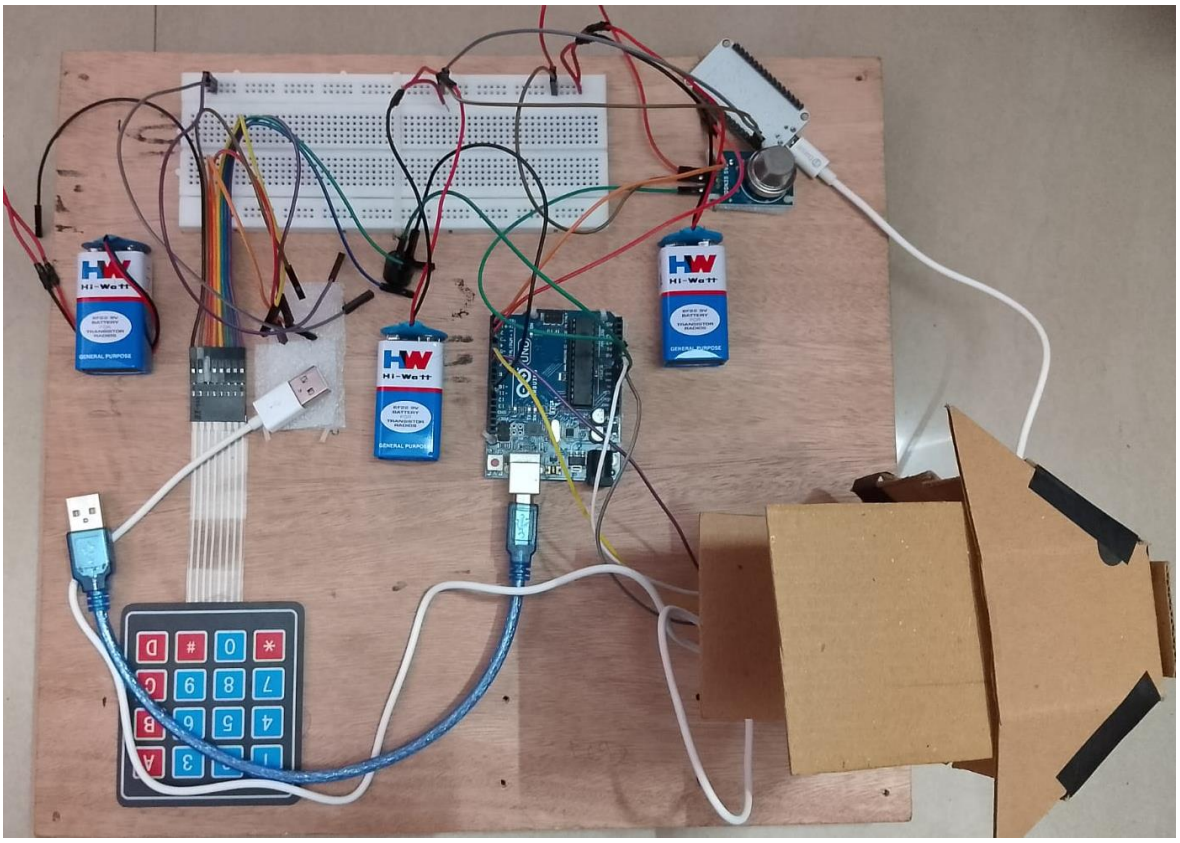
Firstly, the hardware components including the sensors (MQ2, UV), Arduino Uno R3, and ESP8266 modules are carefully selected and deployed throughout the facility. These sensors are strategically placed in areas prone to potential explosion risks, such as near pipelines, storage tanks, and processing equipment.

Once the hardware is in place, the software components are developed and integrated. This includes writing code for data collection, preprocessing, analysis, and alert generation. The data collected by the sensors is transmitted to the central server using the ESP8266 module, where it undergoes preprocessing to remove noise and anomalies. Machine learning algorithms, particularly linear regression models, are then applied to analyze the data and identify potential explosion risks based on historical patterns and current readings.

CHAPTER 5

OUTPUTS

5.1 OUTPUT:



5.2 SECURITY MODEL:

The security model for the home automation system prioritizes safeguarding user privacy, data integrity, and system accessibility. Central to this model is robust authentication, employing methods like password-based, biometric, or multi-factor authentication to verify user identities. Encryption techniques are then employed to secure communication between the mobile application and the Arduino UNO, as well as between the Arduino UNO and external devices, ensuring confidentiality of transmitted data. Authorization mechanisms are implemented to control user access, with role-based access control (RBAC) restricting privileges to sensitive functionalities and data. Secure communication protocols like HTTPS or MQTT with TLS are utilized to encrypt data transmission over the internet, mitigating eavesdropping and man-in-the-middle attacks. Regular updates to firmware and software components address known vulnerabilities, while monitoring and logging mechanisms enable detection of suspicious activities and timely response to security incidents. Through the integration of these components, the security model ensures the home automation system's resilience against potential threats, safeguarding both user information and system integrity.

Secure communication protocols like HTTPS or MQTT with TLS are utilized to encrypt data transmission over the internet, mitigating eavesdropping and man-in-the-middle attacks. Regular updates to firmware and software components address known vulnerabilities, while monitoring and logging mechanisms enable detection of suspicious activities and timely response to security incidents. Through the integration of these components, the security model ensures the home automation system's resilience against potential threats, safeguarding both user information and system integrity.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In conclusion, the IoT-based explosion risk detection system represents a significant advancement in safety measures within the oil and gas industry. Through the integration of sophisticated sensor technology, real-time data analysis, and prompt alert mechanisms, this system has demonstrated its potential to mitigate potential hazards and enhance operational safety.

One of the key strengths of the system lies in its ability to continuously monitor critical environmental parameters such as gas concentrations, temperature, and UV levels. By leveraging advanced machine learning algorithms, the system can analyze large volumes of data to identify patterns and anomalies indicative of potential explosion risks. This predictive capability allows for early detection of hazards, enabling operators to take proactive measures to prevent accidents.

The implementation of a user-friendly monitoring dashboard further enhances the system's effectiveness by providing operators with real-time data visualization and historical trends. This enables operators to make informed decisions and respond promptly to any detected risks.

While the system offers significant benefits in terms of safety and operational efficiency, there are also areas for further improvement. Future enhancements could include the integration of additional sensor technologies, the development of more advanced machine learning algorithms, and the implementation of predictive maintenance strategies to further reduce the risk of incidents.

6.2 FUTURE WORK

Future work for the explosion risk detection system in the oil and gas industry presents several avenues for further enhancement and refinement. One key area of focus is the integration of advanced sensor technologies to improve the accuracy and scope of hazard detection. This includes the incorporation of additional sensors capable of detecting a wider range of gases and environmental parameters, as well as the implementation of advanced data fusion techniques to combine data from multiple sources for more comprehensive risk assessment.

Moreover, there is a need to explore the integration of emerging technologies such as artificial intelligence (AI) and edge computing to enhance the system's predictive capabilities. AI algorithms can analyze large volumes of data in real-time to identify complex patterns and trends, enabling more accurate prediction of potential explosion risks. Edge computing can also be leveraged to perform data processing and analysis closer to the source of data generation, reducing latency and improving system responsiveness.

Another important aspect of future work involves the development of predictive maintenance capabilities for the system infrastructure. By implementing predictive maintenance algorithms, the system can proactively identify and address potential hardware failures or malfunctions before they occur, ensuring continuous operation and minimizing downtime.

Additionally, efforts can be directed towards enhancing the user interface and dashboard functionalities to provide operators with more intuitive and interactive tools for monitoring and managing the system.

REFERENCES

1. Ajagbe, S. A., Adeaga, O. A., & Alabi, O. O. (2024). Design and development of Arduino UNO-based automation home system using the internet of things. [Provide Source if Available]

2. Aalsalem, M. Y., Khan, W. Z., & Gharibi, W. (2017). An intelligent oil and gas well monitoring system based on the Internet of Things. [Provide Source if Available]

3. Khan, W. Z., Aalsalem, M. Y., & Khan, M. K. (2017). A reliable Internet of Things-based architecture for the oil and gas industry. [Provide Source if Available]

4. Aalsalem, M. Y., Khan, W. Z., Gharibi, W., & Khan, M. K. (2018). Wireless Sensor Networks in the oil and gas industry: Recent advances, taxonomy, requirements, and open challenges. [Provide Source if Available]

5. Nguyen, T., Gosine, R. G., & Warrian, P. (2020). A systematic review of big data analytics for the oil and gas industry 4.0. [Provide Source if Available]

6. Sharma, S. K., Rani, A., Bakhariya, H., & Kumar, R. (2024). The Role of IoT in Optimizing Operations in the Oil and Gas Sector: A Review. [Provide Source if Available]

7. Subramanian, M. A., & Selvam, N. (2020). Gas Leakage Detection System using IoT with integrated notifications using Pushbullet - A Review. [Provide Source if Available]

8. Ijiga, O. E., Malekian, R., & Chude-Okonkwo, U. A. K. (2020). Enabling emergent configurations in the industrial Internet of Things for oil and gas explorations: A survey. [Provide Source if Available]

APPENDIX

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <SoftwareSerial.h>
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);
SoftwareSerial mySerial(0, 1);

const int trigPin = 7;
const int echoPin = 6;
const int mqPin = A0;
const int ledPin = 12;
const int buzzerPin = 11;
bool keyPadClicked = false;

void setup() {
  Serial.begin(9600);
  delay(2000);
  Serial.println("Process Started...");

  mySerial.begin(9600);

  lcd.init();
  lcd.backlight();

  pinMode(ledPin, OUTPUT);
  pinMode(buzzerPin, OUTPUT);
```

```
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
    pinMode(mqPin, INPUT);
}

void loop() {
    int overallStatus = OilSensor() && GasSensor();

    if (overallStatus) {
        ledGood();
        buzzerGood();
        lcdGood();
    } else {
        ledBad();
        buzzerBad();
        lcdBad();
        sendSerialAlert();
        receiveKeyPad();
        while(keyPadClicked){
          delay(5000);
        }
    }

    sendSerialData();
    delay(1000);
}

bool OilSensor() {
    digitalWrite(trigPin, LOW);
```

```
    delayMicroseconds(2);

    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);
    int distance = duration * 0.034 / 2;
    return distance <= 6;
}

bool GasSensor() {
    int gasValue = analogRead(mqPin);
    return gasValue <= 100;
}

void ledGood() {
    static unsigned long previousMillis = 0;
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= 2000) {
        digitalWrite(ledPin, !digitalRead(ledPin));
        previousMillis = currentMillis;
    }
}

void ledBad() {
    static unsigned long previousMillis = 0;
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= 500) {
```

```
        digitalWrite(ledPin, !digitalRead(ledPin));
        previousMillis = currentMillis;
    }
}

void buzzerGood() {
    tone(buzzerPin, 0);
}

void buzzerBad() {
    tone(buzzerPin, 1000);
}

void lcdGood() {
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("Status: GOOD");
}

void lcdBad() {
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("Status: BAD");
}

void lcdInvalid() {
    lcd.clear();
    lcd.setCursor(0, 1);
    lcd.print("Invalid Key...");
```

```
}

void lcdStop() {
   lcd.clear();
   lcd.setCursor(0, 1);
   lcd.print("Status: STOPPED");
}

void sendSerialData() {
   StaticJsonDocument<128> sendData;
   sendData["informationType"] = "data";
   sendData["oil"]["oilStatus"] = OilSensor();
   sendData["gas"]["gasStatus"] = GasSensor();
   sendData["timestamp"] = millis();

   String sendJsonString;
   serializeJson(sendData, sendJsonString);
   mySerial.println(sendJsonString);
}

void sendSerialAlert() {
   mySerial.println("{\"informationType\":\"alert\"}");
}

void receiveKeyPad() {
   if (mySerial.available() > 0) {
      StaticJsonDocument<128> receiveData;
      char receiveJsonBuffer[128];
      mySerial.readBytesUntil('\n', receiveJsonBuffer, sizeof(receiveJsonBuffer));
```

```
        DeserializationError error = deserializeJson(receiveData, receiveJsonBuffer);

        if (error) {
            Serial.println("Failed to parse JSON...");
            Serial.println(error.c_str());
        }

        if (strcmp(receiveData["informationType"], "key") == 0) {
            keyPadClicked = false;
        } else {
            lcdInvalid();
            keyPadClicked = true;
        }
    }
}
```

Appendix 2: Backend API Documentation

```
import express from "express";
import * as controllers from "../controllers/login";

const router=express.Router();

router.post("/login",controllers.checkLogin);

router.post("/session",controllers.checkSession);

router.post("/cookie",controllers.checkCookie);

router.post("/gas",controllers.gasData);

router.post("/oil",controllers.oilData);

router.post("/employeeStatus",controllers.employeeStatus);

router.post("/employeeManagement",controllers.employeeManagement);
```

```
router.post("/attendance",controllers.attendance);

router.post("/createDate",controllers.createDate);

router.post("/fetchEvacuator",controllers.fetchEvacuator);

router.post("/evacuator",controllers.evacuator);

router.post("/logout",controllers.logout);

export default router;

import express from "express";
import * as controllers from "../controllers/login";

const router=express.Router();

router.post("/login",controllers.checkLogin);

router.post("/session",controllers.checkSession);

router.post("/cookie",controllers.checkCookie);

router.post("/gas",controllers.gasData);

router.post("/oil",controllers.oilData);

router.post("/employeeStatus",controllers.employeeStatus);

router.post("/employeeManagement",controllers.employeeManagement);

router.post("/attendance",controllers.attendance);

router.post("/createDate",controllers.createDate);

router.post("/fetchEvacuator",controllers.fetchEvacuator);import { NextFunction,
Request, RequestHandler, Response } from "express";
import loginModel from "../models/loginModel";
import { SessionData } from "express-session";
import gasDataModel from "../models/gasDataModel";
import oilDataModel from "../models/oilDataModel";
import employeeDataModel from "../models/employeeDataModel";
import filterDataModel from "../models/filterDataModel";
```

```
import employeeDataInterface from "../models/employeeDataInterface";

interface MySessionData extends SessionData{
   user?:string;
   isLogged?:boolean;
   role?:string;
}

export const
checkLogin:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>
{
   const myindustryid=req.body.industryid;
   const mypassword=req.body.password;
   const myrole=req.body.role;
   try{
      const cred=await loginModel.find({industryid:myindustryid}).exec();
      if(cred.length===0){
         res.status(404).json({error:"Username not found"});
      }
      else{
         if(cred[0].password!==mypassword){
            res.status(406).json({error:"Invalid password"});
         }
         else{
            if(cred[0].role!==myrole){
               res.status(407).json({error:"Invalid role"});
            }
            else{
               (req.session as MySessionData).user=myindustryid;
               (req.session as MySessionData).isLogged=true;
               (req.session as MySessionData).role=myrole;

res.cookie("loginCookie_ERMS",myindustryid,{maxAge:36000000,httpOnly:false})
;
               res.status(200).json({message:"Login successful"});
            }
         }
      }
   }
   catch(error){
      next(error);
   }
}
```

```
export const
checkSession:RequestHandler=async(req:Request,res:Response,next:NextFunction)=
>{
   try{
      const isSession=(req.session as MySessionData).isLogged;
      if(isSession){
         res.status(200).json({message:"Session found",data:{
            username:(req.session as MySessionData).user,
            role:(req.session as MySessionData).role,
         }});
      }
      else{
         res.status(404).json({message:"Session not found"});
      }
   }
   catch(error){
      next(error)
   }
}

export const
checkCookie:RequestHandler=async(req:Request,res:Response,next:NextFunction)=
>{
   try{
      if(req.cookies["loginCookie_ERMS"]){
         (req.session as MySessionData).user=req.cookies["loginCookie_ERMS"];
         (req.session as MySessionData).isLogged=true;
         res.status(200).json({message:"Cookie found",data:{
            username:(req.session as MySessionData).user,
            role:(req.session as MySessionData).role,
         }});
      }
      else{
         res.status(404).json({message:"Cookie not found"});
      }
   }
   catch(error){
      next(error)
   }
}

export const
gasData:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
   try{
```

```
        const data=await gasDataModel.find({}).sort({timeStamp:"desc"})
        res.status(200).json({data:data[0]})
    }
    catch(error){
        next(error)
    }
}

export const
oilData:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
    try{
        const data=await oilDataModel.find({}).sort({timeStamp:"desc"})
        res.status(200).json({data:data[0]})
    }
    catch(error){
        next(error)
    }
}

export const
employeeStatus:RequestHandler=async(req:Request,res:Response,next:NextFunction
)=>{
    const passValue:filterDataModel={}
    if(req.body){
        if(req.body.employeeID) passValue.employeeID=req.body.employeeID
        else{
            if(req.body.sectorNo) passValue.sectorNo=req.body.sectorNo
            if(req.body.attendance==="Present") passValue.attendance=true
            else if(req.body.attendance==="Absent") passValue.attendance=false
            if(req.body.evacuated==="Yes") passValue.evacuated=true
            else if(req.body.evacuated==="No") passValue.evacuated=false
            if(req.body.date) passValue.date=req.body.date
        }
    }
    try{
        const data=await employeeDataModel.find(passValue).exec()
        res.status(200).json({data:data})
    }
    catch(error){
        next(error)
    }
}

export const
```

```
employeeManagement:RequestHandler=async(req:Request,res:Response,next:NextF
unction)=>{
   const passValue:filterDataModel={}
   if(req.body){
      if(req.body.employeeID) passValue.employeeID=req.body.employeeID
      else{
         if(req.body.sectorNo) passValue.sectorNo=req.body.sectorNo
         if(req.body.attendance==="Present") passValue.attendance=true
         else if(req.body.attendance==="Absent") passValue.attendance=false
         if(req.body.evacuated==="Yes") passValue.evacuated=true
         else if(req.body.evacuated==="No") passValue.evacuated=false
         if(req.body.date) passValue.date=req.body.date
      }
   }
   try{
      const data=await employeeDataModel.find(passValue).exec()
      res.status(200).json({data:data})
   }
   catch(error){
      next(error)
   }
}

export const
attendance:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
   const {id,value,date}=req.body
   try{
      const data=await
employeeDataModel.updateOne({employeeID:id,date:date},{attendance:value})
      res.status(200).json({data:"Updated",status:data})
   }
   catch(error){
      next(error)
   }
}

export const
createDate:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
   const date=req.body.date
   try{
      const data=await employeeDataModel.find({date:date})
      if(data.length>0){
         res.status(409).json({message:"Register already exists!"})
      }
```

```
        else{
            const newData=await employeeDataModel.find({date:"2024-01-01"})
            const newArr:employeeDataInterface[]=newData.map((item)=>{
                return{
                    employeeID:item.employeeID,
                    employeeName:item.employeeName,
                    sectorNo:item.sectorNo,
                    attendance:false,
                    evacuated:false,
                    date:date,
                }
            })
            const create=await employeeDataModel.insertMany(newArr)
            res.status(200).json({message:"Register created
successfully",data:create.length})
        }
    }
    catch(error){
        next(error)
    }
}

export const
fetchEvacuator:RequestHandler=async(req:Request,res:Response,next:NextFunction
)=>{
    const {id,date}=req.body
    try{
        const data=await employeeDataModel.find({employeeID:id,date:date}).exec()
        res.status(200).json({status:data[0]})
    }
    catch(error){
        next(error)
    }
}

export const
evacuator:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
    const {id,value,date}=req.body
    try{
        const data=await
employeeDataModel.updateOne({employeeID:id,date:date},{evacuated:value})
        res.status(200).json({updateStatus:"Updated",data:data,status:(!value)})
    }
    catch(error){
```

```
        next(error)
    }
}

export const
logout:RequestHandler=async(req:Request,res:Response,next:NextFunction)=>{
    try{
        req.session.destroy((err)=>{
            if(err) res.status(500).json({message:"Error logging out"})
        })
        res.clearCookie("loginCookie_ERMS")
        res.status(200).json({message:"Logged out successfully"})
    }
    catch(error){
        next(error)
    }
}

router.post("/evacuator",controllers.evacuator);

router.post("/logout",controllers.logout);

export default router;
```