# 

[**IoT-Based Warehouse Sensor Network Documentation**](#_spqnd970eobx) **2**

[Introduction](#_xsrdx9e0ce21) 2

[Project Overview](#_7yjj7jip1e8b) 2

[System Architecture](#_vbgyxrpmbpom) 2

[Components Used:](#_cx2cmcmn1a8j) 2

[Hardware Setup](#_u3gnnmkv2anu) 2

[Arduino Uno (Atmega328p):](#_u4q539ptbzpi) 2

[ESP32](#_enxwpvaa1lwx) 3

[Flame Sensor](#_hr8wrqpurnz) 3

[Thermal Sensor](#_se98sgllcha) 3

[Software Setup](#_rsurkwhtflt5) 3

[Microchip Studio and AVR Assembly](#_7nw2zxkbn4wf) 3

[Arduino IDE and C++ Programming](#_l9mqlnt44xlq) 3

[MQTT Dash Mobile App](#_ectw1l1xcsfz) 4

[Communication Protocol](#_s4w24vasz23j) 4

[UART Serial Communication](#_mw2y4gao2hh6) 4

[Tx/Rx Pins Configuration](#_39gvhbs0e6p1) 4

[Implementation Details](#_tirf0md807jc) 4

[Arduino Uno Implementation](#_u3zyi69u19zx) 4

[ESP32 Implementation](#_3on267wl98nl) 4

[MQTT Broker (Mosquitto) Settings:](#_z15mfz12s442) 4

[IoT Module Programming}](#_cru0riwzn4u9) 6

[Operation](#_5uym91wp81fl) 6

[Sensor Data Acquisition](#_fy5jlzvdcibv) 6

[Troubleshooting](#_s5clgu9w2uzt) 7

[Common Issues](#_xg5ya6zcmadb) 7

[Debugging Techniques](#_r9znv9rmnrkp) 7

[Conclusion](#_6nsha7uars71) 7

[Project Summary](#_z24lnxoirdqn) 7

[Future Improvements](#_89s4na2l11e2) 7

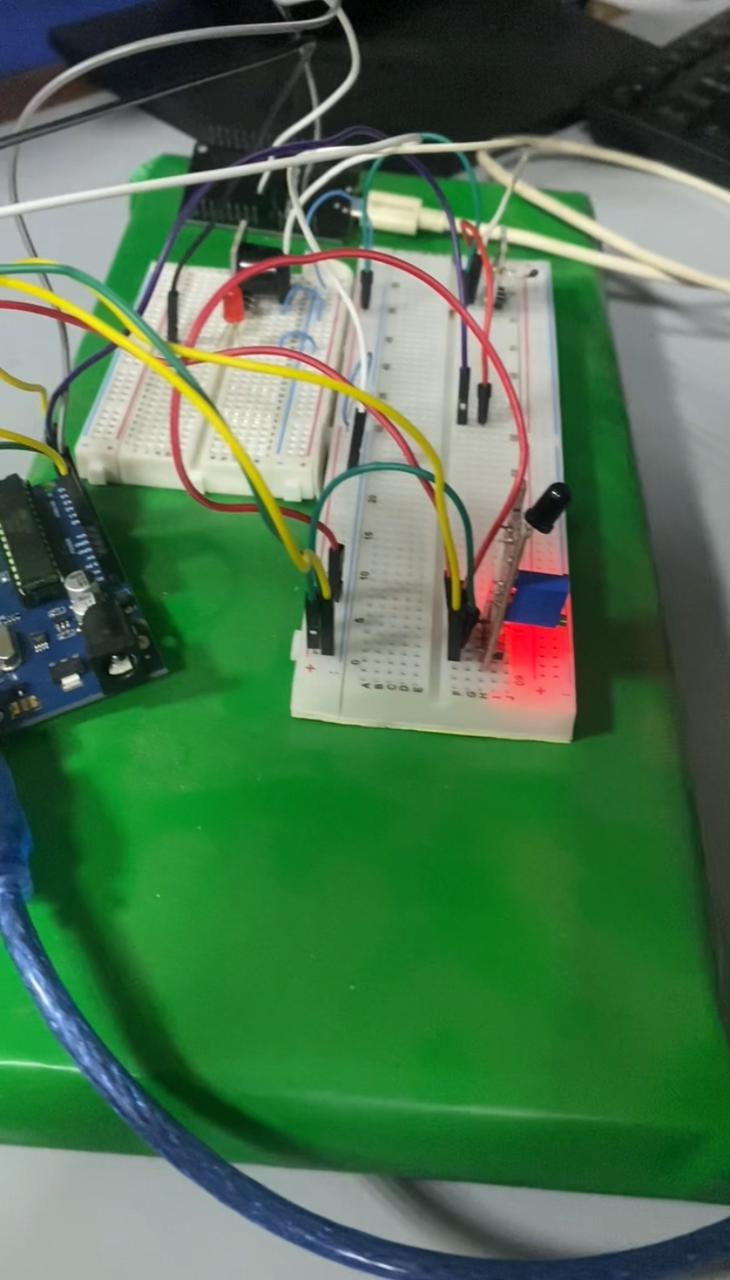
# IoT-Based Warehouse Sensor Network Documentation

## Introduction

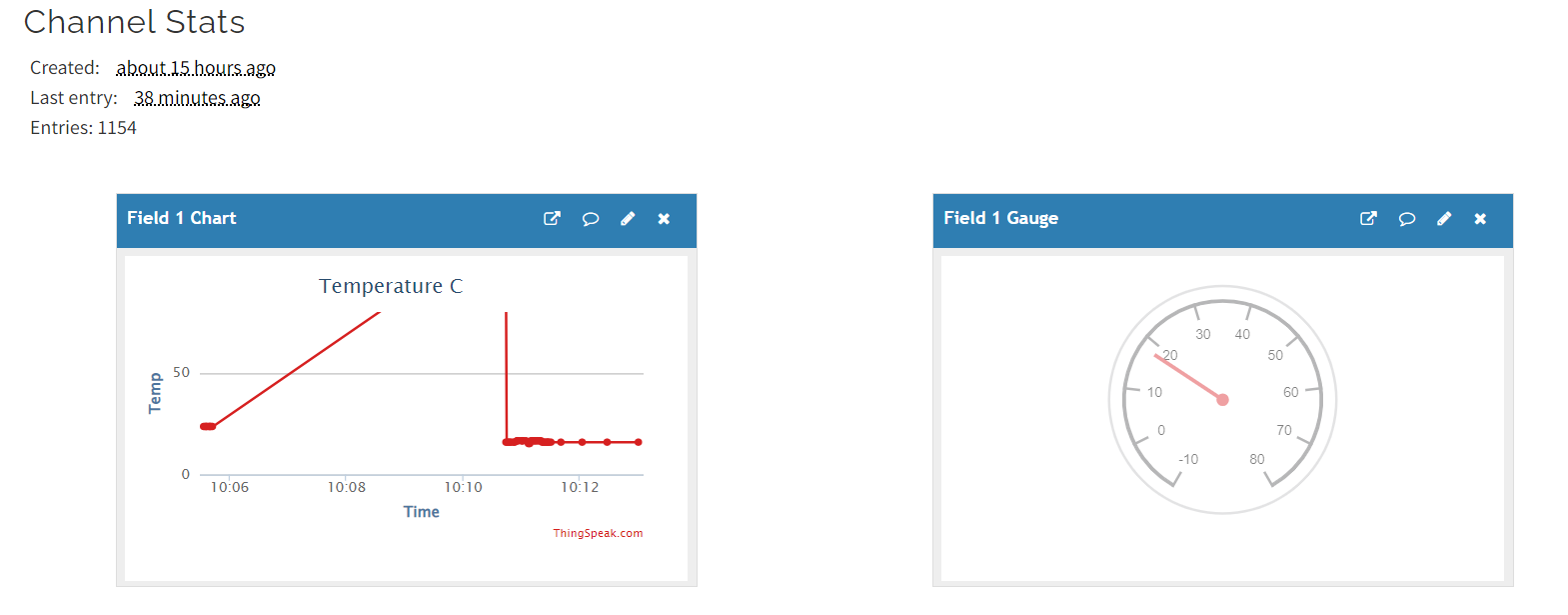
### Project Overview

The "IoT-Based Warehouse Sensor Network" project aims to create a smart warehouse monitoring system using Arduino Uno (Atmega328p) and ESP32 microcontrollers. The system incorporates flame and thermal sensors to detect potential hazards in the warehouse environment. Users can control and monitor the system through a mobile app named "MQTT Dash," which communicates with the ESP32 module.

### Project Hardware Pictures:







### 

### System Architecture

The system consists of Arduino Uno as the main controller, ESP32 as the IoT module, flame, and thermal sensors for data acquisition. The mobile app communicates with the ESP32 module through the MQTT protocol.

#### Components Used:

* Arduino Uno (Atmega328p)
* ESP32
* Flame Sensor
* Thermal Sensor
* Mobile App: MQTT Dash

## Hardware Setup

### Arduino Uno (Atmega328p):

The flame and thermal sensors are connected to the appropriate pins on the Arduino Uno. Ensuring proper power supply and ground connections. Additionally, the light and buzzer are based on the specified temperature thresholds.

### ESP32

The ESP32 is connected to the Arduino Uno using UART serial communication. Connections are established between the Tx/Rx (Transmitter and receiver) pins on both devices. A secure physical connection is ensured to enable reliable data transfer.

### Flame Sensor

The flame sensor is connected to the Arduino Uno. The pin configurations are as follows:

### Thermal Sensor

The thermal sensor is connected to the Arduino Uno, following the recommended pin configurations. The functionality to turn on a light at a temperature of about 26 degrees Celsius and activate a buzzer when the temperature reaches 30 degrees Celsius is integrated in the Arduino Uno firmware. The pin configurations are as follows:

## Software Setup

### Microchip Studio and AVR Assembly

The firmware of the project for Arduino Uno is created using AVR assembly language in Microchip Studio.

### Arduino IDE and C++ Programming

To Program the IoT module (ESP32) C++ is used in the Arduino IDE.

### MQTT Dash Mobile App

The MQTT app is installed on a mobile phone to remotely control the system. The app is configured to connect to the ESP32 module using the appropriate MQTT settings.

## Communication Protocol

### UART Serial Communication

UART serial communication is used between Arduino Uno and ESP32 for data exchange.

### Integration with Firebase and Data Storage:

Firebase Integration:

The IoT-based warehouse sensor network project is seamlessly integrated with Firebase, a robust cloud-based platform. Firebase serves as a central repository for storing and managing crucial data generated by the system, ensuring secure and accessible storage for historical records and real-time updates.

Data Storage in CSV Format:

To enhance data traceability and analysis, the system is configured to store sensor readings and relevant information in a Comma-Separated Values (CSV) format. This standardized format simplifies data processing and enables compatibility with various analysis tools.

Automated Data Uploads:

The project includes an automated process that regularly uploads data to Firebase in CSV format. This ensures that historical records and real-time sensor readings are consistently stored in the cloud. The automated uploads facilitate easy retrieval and analysis of data, contributing to comprehensive insights into warehouse conditions over time.

Advantages of Firebase Integration:

1. Real-time Accessibility:

- Firebase integration allows authorized users to access sensor data in real-time from any location. This enhances the system's transparency and responsiveness.

2. Scalability and Reliability:

- Firebase offers a scalable and reliable cloud infrastructure, accommodating the growing volume of data generated by the warehouse sensor network over time.

3. Data Analysis and Visualization:

- By storing data in CSV format and uploading it to Firebase, users gain the flexibility to perform in-depth data analysis and visualization. This capability aids in identifying patterns, trends, and potential areas for improvement in warehouse conditions.

4. Historical Record Keeping:

- Historical sensor readings are securely stored in Firebase, creating a comprehensive record of the warehouse environment's past states. This historical data is invaluable for audits, compliance, and retrospective analysis.

This integration with Firebase and the implementation of CSV data storage and uploads significantly enhance the project's capabilities, providing users with advanced tools for data analysis, historical tracking, and real-time monitoring of the warehouse sensor network.

## Implementation Details

### 

### ESP32 Implementation

The MQTT communication parameters for connecting to the Mosquitto Broker app on the laptop are as follows:

### MQTT Broker (Mosquitto) Settings:

IP Address: <Laptop IP>

Port: 1833 (default port used by Mosquitto)

The ESP32, acting as an MQTT client, connects to the Mosquitto Broker using the configured IP address and port. It subscribes to a specific MQTT topic where it expects commands from the Arduino Uno:

Command Topic:

Topic: warehouse/commands

The ESP32 periodically publishes the buzzer value to a different MQTT topic, indicating whether the button is on or off:

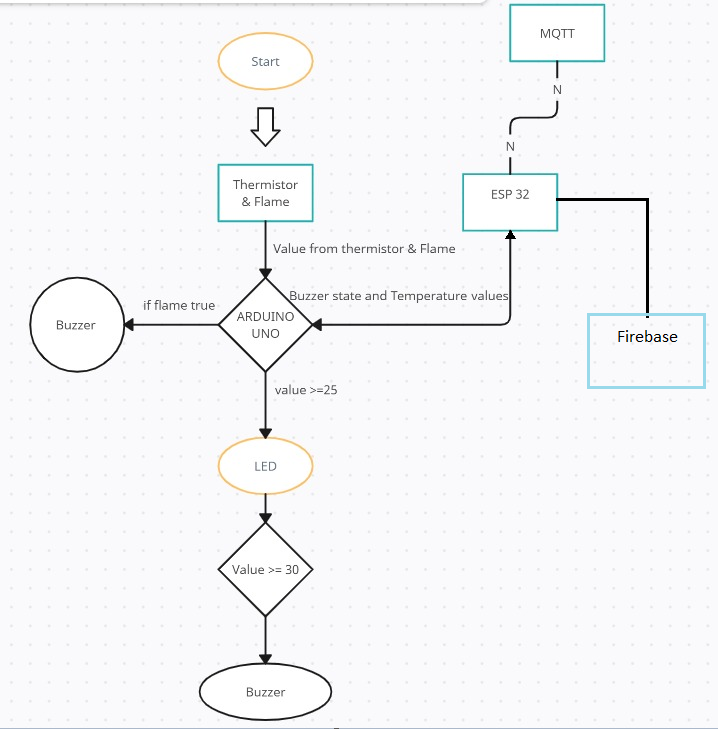
Buzzer Status Topic:

Topic: warehouse/buzzer\_status

When the ESP32 receives a command from the subscribed MQTT topic (warehouse/commands), it processes the command and sends the corresponding buzzer value to the Arduino Uno via UART communication. Specific actions can be defined based on the received command, such as turning on or off the buzzer.

### IoT Module Programming

C++ code is written in Arduino IDE for the IoT module, defining MQTT communication parameters and interaction with the mobile app.



## Operation

### Sensor Data Acquisition

The process by which the Arduino Uno reads data from flame and thermal sensors involves a series of steps. Firstly the flame sensor is connected to specific pins on the Arduino Uno, ensuring appropriate power and ground connections. The Arduino Uno then initializes the flame sensor for communication and proceeds to read the analog signal from the flame sensor using one of its analog input pins.

The obtained analog value is compared against a predefined threshold. If the analog value exceeds this threshold, it signifies the presence of a flame. Subsequently, the Arduino Uno triggers the logic to turn on a light, connected to one of its output pins, providing a visual indication of the detected flame.

Similarly, for the thermal sensor, the initialization process involves connecting the sensor to specific pins on the Arduino Uno and ensuring proper power and ground connections. The thermal sensor is then ready for communication, and the Arduino Uno reads the analog signal from the thermal sensor using one of its analog input pins.

The analog value obtained from the thermal sensor corresponds to the temperature. The Arduino Uno compares this temperature value against predefined thresholds. If the temperature exceeds a certain threshold, for example, 30 degrees Celsius, the Arduino Uno triggers the logic to activate a buzzer, connected to one of its output pins. Additionally, based on temperature readings, the Arduino Uno can trigger the logic to turn on a light as a secondary visual indicator of the temperature status.

To ensure safety and accuracy, the system incorporates a careful definition of temperature thresholds for flame detection and temperature triggering.

### Data Transfer between Arduino Uno and ESP32

UART communication between the Arduino Uno and ESP32 involves the bidirectional exchange of serial data through the Tx (transmit) and Rx (receive) pins. To facilitate this communication, both microcontrollers are configured to use the same baud rate, determining the speed at which data is transmitted and received.

When the Arduino Uno wishes to send data to the ESP32, it utilizes the Serial library to write data to its Tx pin, while the ESP32 receives the data on its Rx pin. Each byte of data transmitted includes start and stop bits to frame the information and synchronize communication.

### Mobile App Control via MQTT Dash

The mobile app is an important interface in the project that allows the users to interact with the alarm system and access real-time temperature information. Through MQTT communication with the ESP32, the mobile app allows controls for managing the alarm, including the light and buzzer, as well as providing continuous temperature updates.

## Conclusion

### Project Summary

In summary, The IoT-based warehouse sensor network stands out for its successful integration of hardware components and microcontrollers, creating a sophisticated monitoring system. Key achievements include the incorporation of flame and thermal sensors. The collaboration between the Arduino Uno and ESP32 microcontrollers ensures efficient data exchange, enabling seamless control and monitoring within the system.

A noteworthy functionality is the real-time alarm system, featuring a light and buzzer, providing immediate visual and auditory alerts in response to flame detection or critical temperature changes. The mobile app, MQTT Dash, serves as an interface, allowing remote control of the alarm system components and real-time temperature monitoring through MQTT communication with the ESP32.

## Project Code:

;==============================================================================

; Push Button and Temperature Sensor Control System

;==============================================================================

;==============================================================================

; Initialization Section

;==============================================================================

; Configure Ports and Pins

; - PB5: Output pin for LED

; - PD2: Input pin for push button

; - PD7: Output pin for buzzer

; - PD6: Output pin for additional LED

.include "m328pdef.inc"

.include "delay\_Macro.inc"

.include "UART\_Macros.inc"

.include "div\_Macro.inc"

.cseg

.def A = r16 ; General-purpose register

.def AH = r17 ; General-purpose register

.def FlameBuzzer = r18 ; Flag for flame buzzer state

.def ThermalBuzzer = r19 ; Flag for thermal buzzer state

.org 0x0000

SBI DDRB, PB5 ; Set PB5 as OUTPUT Pin

CBI PORTB, PB5 ; LED OFF

CBI DDRD, PD2 ; Set PD2 as INPUT pin

SBI DDRD, PD7 ; Set PD7 as OUTPUT Pin

CBI PORTD, PD7 ; Buzzer OFF

SBI DDRD, PD6 ; Set PD6 as OUTPUT Pin

CBI PORTD, PD6 ; Additional LED OFF

; ADC Configuration

LDI A, 0b11000111 ; [ADEN ADSC ADATE ADIF ADIE ADIE ADPS2 ADPS1 ADPS0]

STS ADCSRA, A

LDI A, 0b01100000 ; [REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0]

STS ADMUX, A ; Select ADC0 (PC0) pin

Serial\_begin ; Initialize UART serial communication

;==============================================================================

; Main Loop Section

;==============================================================================

loop:

delay 1000

; Read thermal value from temperature sensor

call loadThermalVal

; Send the received value to ESP via UART

Serial\_writeReg\_ASCII AH

; Compare thermal value with the desired threshold

cpi AH, 136

brlo OnThermalOff

; Temperature is hot

SBI PORTB, PB5 ; LED ON

SBI PORTD, PB6 ; Additional LED ON

; Check if flame is detected

SBIC PIND, PD2

rjmp IfFlameOff

; Flame is detected

SBI PORTB, PB5 ; LED ON

rjmp EndLoop

OnThermalOff:

; Temperature is cold

CBI PORTB, PB5 ; LED OFF

CBI PORTD, PB6 ; Additional LED OFF

rjmp FlameSection

IfFlameOff:

; Flame is not detected

CBI PORTB, PB5 ; LED OFF

EndLoop:

Serial\_writeChar '-'

; Check buzzer states and update accordingly

cpi FlameBuzzer, 1

breq onBuzzerOn

cpi ThermalBuzzer, 1

breq onBuzzerOn

Serial\_writeChar '0'

rjmp onBuzzerOff

onBuzzerOn:

; Buzzer is ON

Serial\_writeChar '1'

SBI PORTD, PD7 ; Buzzer ON

rjmp ReadFromEsp

onBuzzerOff:

;==============================================================================

; Read Input from ESP Section

;==============================================================================

ReadFromEsp:

LDI r16, 0

; Check UART serial input buffer for any incoming data and place in r16

Serial\_read

; If there is no data received in UART serial buffer (r16 == 0),

; then don't send it to UART

CPI r16, 0

BREQ skip\_UART

; Check for specific commands

CPI r16, 'B'

breq externalTurnBuzzerOff

; Default action: Turn Buzzer ON

SBI PORTD, PD7 ; Buzzer ON

LDI ThermalBuzzer, 1

LDI FlameBuzzer, 1

; Continue to loop

rjmp skip\_UART

externalTurnBuzzerOff:

; External command to turn Buzzer OFF

CBI PORTD, PD7 ; Buzzer OFF

LDI ThermalBuzzer, 0

LDI FlameBuzzer, 0

skip\_UART:

; Continue looping

rjmp loop

;==============================================================================

; Subroutine to Load Thermal Value

;==============================================================================

loadThermalVal:

LDS A, ADCSRA ; Start Analog to Digital Conversion

ORI A, (1 << ADSC)

STS ADCSRA, A

wait:

LDS A, ADCSRA ; Wait for ADC conversion to complete

sbrc A, ADSC

rjmp wait

LDS A, ADCL ; Must Read ADCL before ADCH

LDS AH, ADCH

ret

## Code documentation:

### Initialization Section

Ports and Pins Configuration

PB5 (LED): Configured as an output pin. Used to control an LED indicating the system state.

PD2 (Push Button): Configured as an input pin. Monitors the state of the push button.

PD7 (Buzzer): Configured as an output pin. Controls a buzzer indicating system activation.

PD6 (Additional LED): Configured as an output pin. Additional LED for visual indication.

### Analog-to-Digital Converter (ADC) Configuration

Configures the ADC for reading temperature values from a sensor connected to ADC0 (PC0).

Initializes the ADC control and multiplexer registers.

### Serial Communication Initialization

Initializes UART communication for serial communication with an ESP module.

### Main Loop Section

Delay: Pauses for 1000 milliseconds.

Read Thermal Value: Calls the loadThermalVal subroutine to read the temperature from the sensor.

### Send Thermal Value to ESP: Sends the temperature value to the ESP module via UART.

Temperature Comparison: Compares the temperature value with a predefined threshold.

If the temperature is above the threshold, turns on LEDs and checks for flame detection.

If the temperature is below the threshold, turns off LEDs and proceeds to the FlameSection.

### Flame Section

Checks if a flame is detected using the state of the push button (PD2).

If a flame is detected, turns on an LED.

End Loop

Writes a '-' character to the UART as a separator.

Checks the state of FlameBuzzer and ThermalBuzzer flags to determine the system state.

If either buzzer is active, writes '1' to the UART; otherwise, writes '0'.

Buzzer Control

onBuzzerOn: Activates the buzzer and sets relevant flags.

onBuzzerOff: Deactivates the buzzer.

Read Input from ESP Section

Reads data from the ESP module via UART.

If 'B' is received, turns off the buzzer.

If other data is received, activates the buzzer and sets flags accordingly.

Subroutine: Load Thermal Value

Initiates an ADC conversion to read the thermal value.

Waits for the conversion to complete.

Reads the ADC values (ADCL and ADCH) to obtain the thermal value.

## Future Improvements

Improving an IoT-based warehouse sensor network project involves enhancing various aspects, including functionality, reliability, and scalability. Here are some potential areas for improvement:

Sensor Accuracy and Redundancy:

Evaluate and upgrade the accuracy of the flame and thermal sensors to enhance the system's reliability in detecting hazards.

Consider implementing redundancy by integrating multiple sensors for each parameter. This helps in cross-verifying readings and increases the system's fault tolerance.

Power Efficiency:

Optimize power consumption, especially for battery-operated devices like the ESP32. Implement sleep modes and power management techniques to extend battery life.

Explore the use of low-power sensor modules or energy-efficient alternatives.

Security Measures:

Implement security features for communication between the Arduino Uno, ESP32, and the mobile app. This could involve encryption protocols and secure communication channels.

Protect sensitive data and credentials used for communication, ensuring the integrity of the system.

User Interface and Experience:

Enhance the user interface of the mobile app (MQTT Dash) to provide a more intuitive and user-friendly experience.

Consider implementing additional features such as historical data visualization, alerts, and notifications for better user interaction.

Adaptive Machine Learning:

Implement adaptive temperature and flame detection thresholds based on historical data analysis. Machine learning algorithms can adjust thresholds dynamically to accommodate changes in the warehouse environment.

Fault Detection and Diagnostics:

Integrate features for fault detection and diagnostics. Implement logging mechanisms to record system events and errors, aiding in troubleshooting and maintenance.

Environmentally Friendly Practices:

Implement eco-friendly practices, such as using energy-efficient components and materials in sensor construction.

Explore the use of recyclable or biodegradable materials in the construction of the sensor devices.

Regulatory Compliance:

Ensure that the project complies with relevant industry standards and regulations related to safety and data privacy.

Stay informed about updates in regulations and standards that may impact the project.

Community and Open Source Contributions:

Share the project as an open-source initiative to encourage collaboration and contributions from the community.

Consider integrating feedback and improvements suggested by the community to enhance the project's robustness and versatility.

Links:

Github: <https://github.com/haiderkumail/Coal-Project>

Linkedin:<https://www.linkedin.com/posts/fullmetaldanish_iot-firebaseintegration-smartwarehousing-activity-7148903630966841344-5Y8w?utm_source=share&utm_medium=member_ios>

Youtube: <https://youtu.be/8l-vbRSl91Y?feature=shared>