Industry-specific intelligent fire management system

Ramco Institute of Technology, Rajapalayam

Project Report

Team ID: PNT2022TMID51070

Industry Mentor: Santoshi

Faculty Mentor: G.Subhashini

TEAM MEMBERS:

Giritharan D
Aravind M
Arumugaviswanath P
Chidamaranathan R

Project Report Format

1. INTRODUCTION
    1. Project Overview
    2. Purpose
2. LITERATURE SURVEY
    1. Existing problem
    2. References
    3. Problem Statement Definition
3. IDEATION & PROPOSED SOLUTION
    1. Empathy Map Canvas
    2. Ideation & Brainstorming
    3. Proposed Solution
    4. Problem Solution fit
4. REQUIREMENT ANALYSIS
    1. Functional requirement
    2. Non-Functional requirements
5. PROJECT DESIGN
    1. Data Flow Diagrams
    2. Solution & Technical Architecture
    3. User Stories
6. PROJECT PLANNING & SCHEDULING
    1. Sprint Planning & Estimation
    2. Sprint Delivery Schedule
    3. Reports from JIRA
7. CODING & SOLUTIONING (Explain the features added in the project along with code)
    1. Feature 1
    2. Feature 2
    3. Database Schema (if Applicable)
8. TESTING
    1. Test Cases
    2. User Acceptance Testing
9. RESULTS
    1. Performance Metrics
10. ADVANTAGES & DISADVANTAGES
11. CONCLUSION
12. FUTURE SCOPE
13. APPENDIX

1. INTRODUCTION

1.1 Project Overview

The goal of the "Industry Specific-Intelligent fire management system" is to prevent unintentional fire mishaps in industries and take the necessary steps to put a stop to any events. A gas sensor, flame sensor, and temperature sensor are all used by the smart fire management system to monitor environmental changes. The sprinklers will turn on right away if a flame is spotted. The model includes a MQ2 gas sensor for detecting methane and propane gases, an IR flame sensor module for detecting flames, and an LM35 temperature sensor for measuring the surroundings. Based on the temperature information and whether any gases are present, the exhaust fans are turned ON. These readings are saved in Cloudant DB, and IBM Watson IOT Platform continuously monitors them and utilising the Nexmo SMS.

1.2 PURPOSE

- Providing a dashboard with a simple administration system and an overview of the user's experience.
- The capability of IoT devices to determine a room's status.
- To activate exhaust fans and sprinklers in the event of an accident.
- To transmit and store the current temperature in the cloud.
- To notify the authorities by SMS if there is a fire accident.

2. LITERATURE SURVEY

2.1. Existing problem

The situation is less than ideal because many buildings lack an automatic alarm system for administrators and authorities as well as a dependable, efficient, affordable, current processing, or feature-rich fire management system. Since they are using antiquated fire protection technologies, the sprinkler system cannot even be activated, and none of them work together to properly prevent false alarms. Applications are also utilised to keep an eye on the entire network.

2.2 References

2.3 Problem Statement Definition

The fire management systems in homes and companies lack features like an automatic alarm system for administrators and authorities and are not very dependable, efficient, or inexpensive. Many structures still have out-of-date fire safety systems that can't even turn on the sprinklers and that interact with one another incorrectly to prevent false alerts. Applications are also used to keep an eye on the entire system.

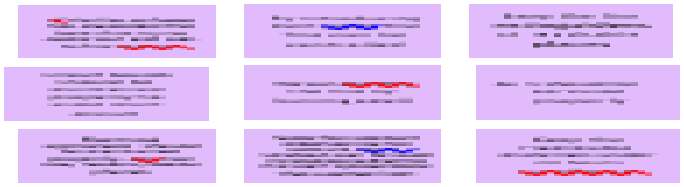
3. IDEATION AND PROPOSED SOLUTION

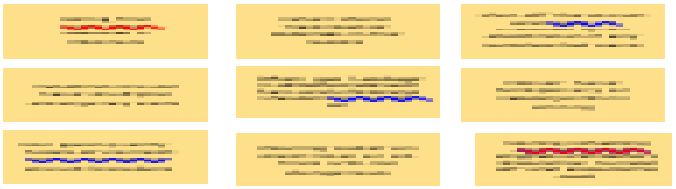
3.1 Empathy Map Canvas

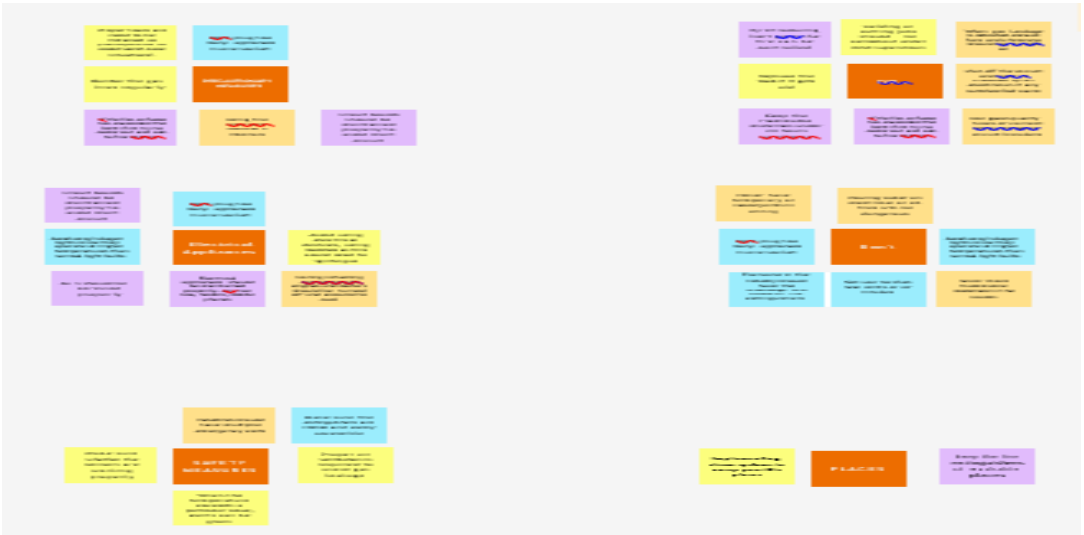
A concise, easy-to-understand image called an empathy map condenses information about a user's activities and viewpoints. It is a useful tool that enables teams to better understand their users. In order to come up with a feasible solution, it's critical to understand both the real problem and the person experiencing it. Through the exercise of building the map, participants learn to consider issues from the user's perspective, including goals and problems.

3.2 Ideation & Brainstorming

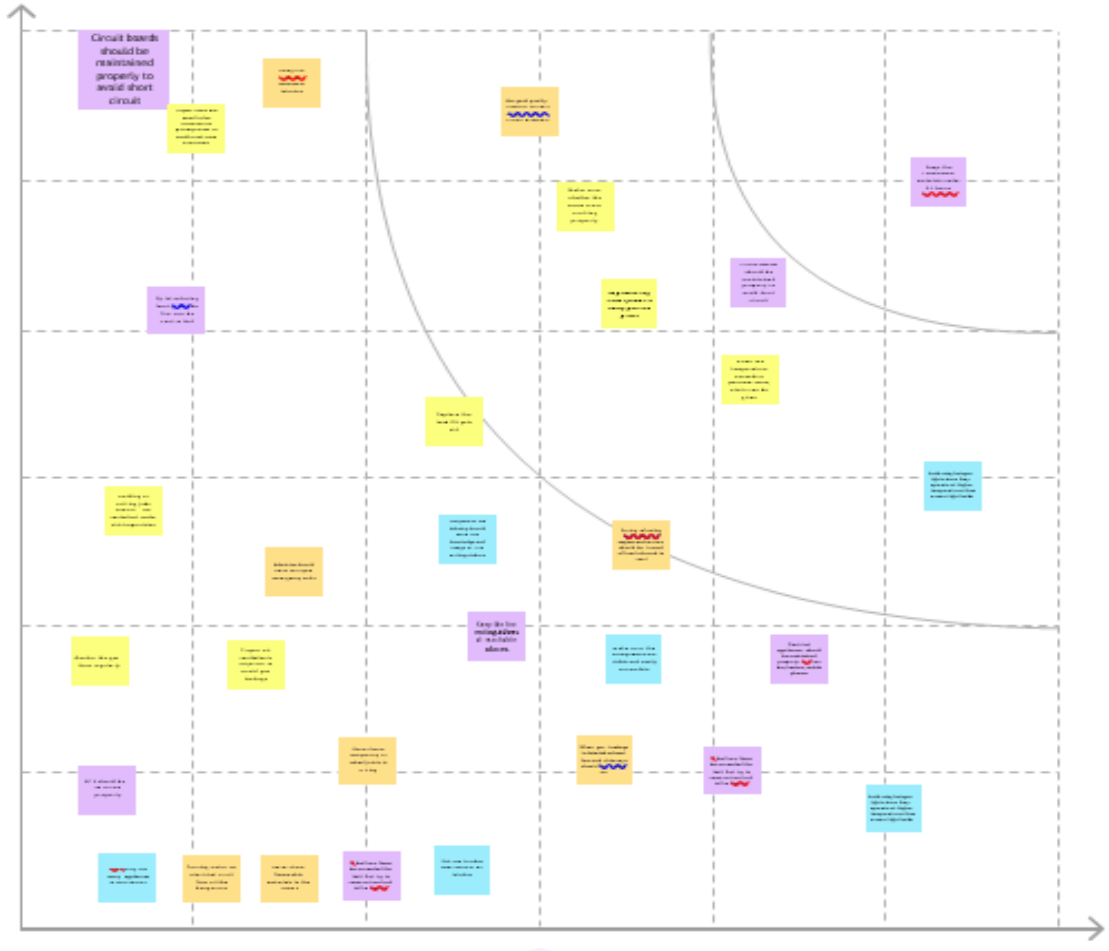
Step 1: Brainstorm, Idea Listing and Grouping:







Step 2: Idea Prioritization

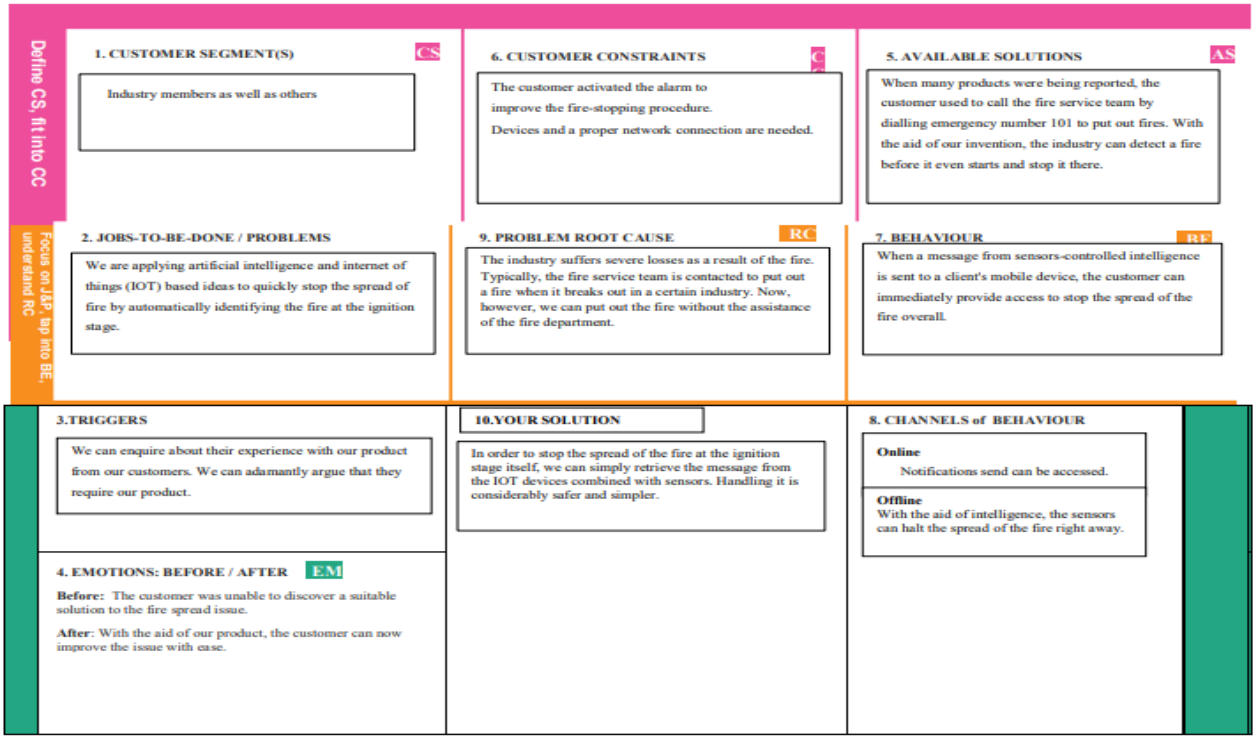


3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To improve the safety management system in industries. Improving the safety management system against the fire incidents in industries.
2.	Idea / Solution description	To implement the fire safety management in industry based on IOT using Arduino uno board with fire detection and fire extinguisher system. And using some sensors (Humidity sensor, Flame sensor, smoke sensor) with GPS tracking system.
3.	Novelty / Uniqueness	An Integrated system of temperature monitoring, gas monitoring, fire detection automatically fire

		extinguisher with accuration of information about locations and response through SMS notification and call.
4.	Social Impact / Customer Satisfaction	It early prevents the accident cost by fire in industries.Nearby locations so maximum extend more accurate reliability.Compatability design integrated system
5.	Business Model (Revenue Model)	This product can be utilized by a industries .this can be thought of as a productive and helpful item as industries great many current rescuing people and machine from the fire accident.
6.	Scalability of the Solution	It is trying to execute this technique as we need to introduce an arduino gadget which was modified with an Arduino that takes received signals from sensors .Easy operatability and maintenance.Required low time for maintain.Cost is reasonable value.

3.4 Problem Solution fit



4.REQUIREMENT ANALYSIS

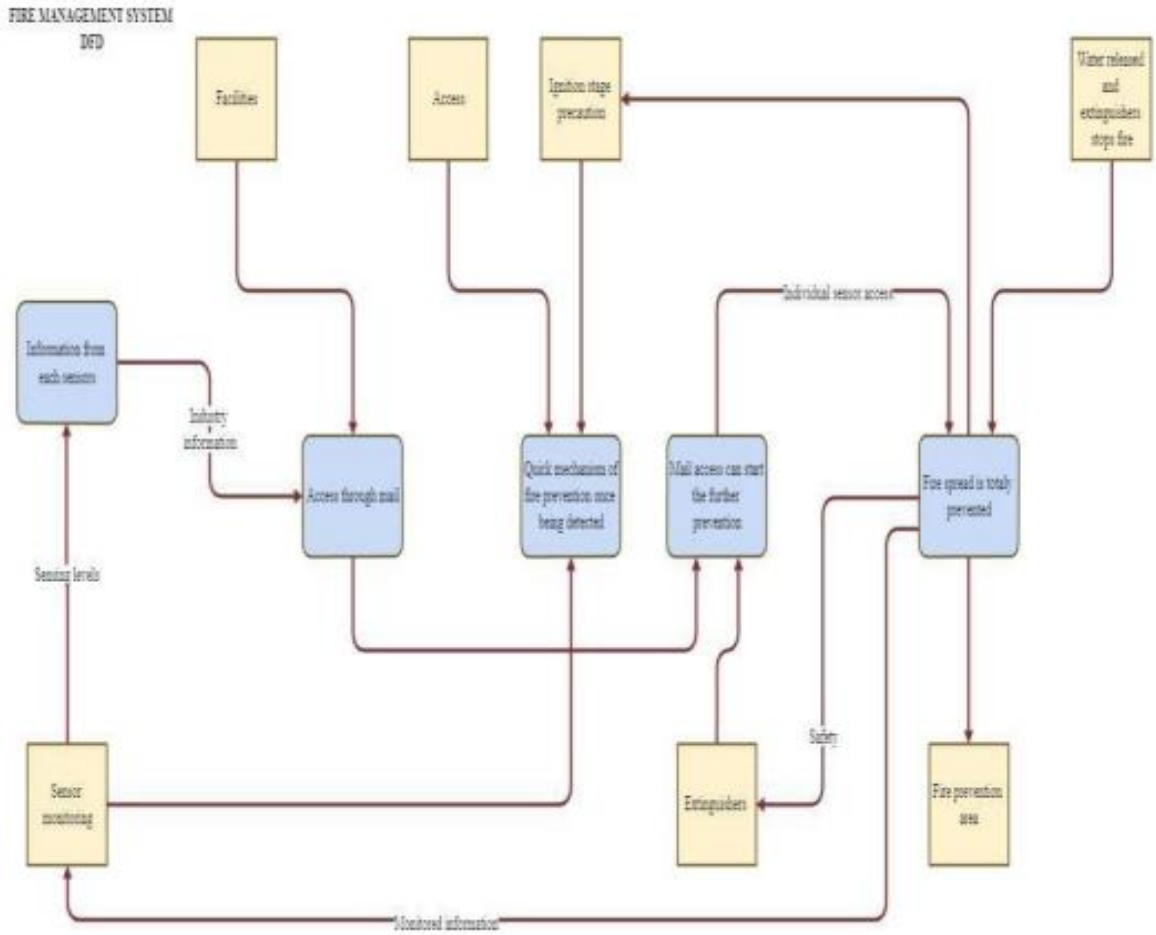
4.1 Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through website or application Registration through Social medias Registration through Linked IN
FR-2	User Confirmation	Verification via Emailor OTP
FR-3	User Login	Login through website or App using the respective username and password
FR-4	User Access	Access the app requirements
FR-5	User Upload	User should be able to upload the data
FR-6	User Solution	Data report should be generated and deliveredto user for every 24 hours
FR-7	User Data Sync	API interface to increase to invoice system

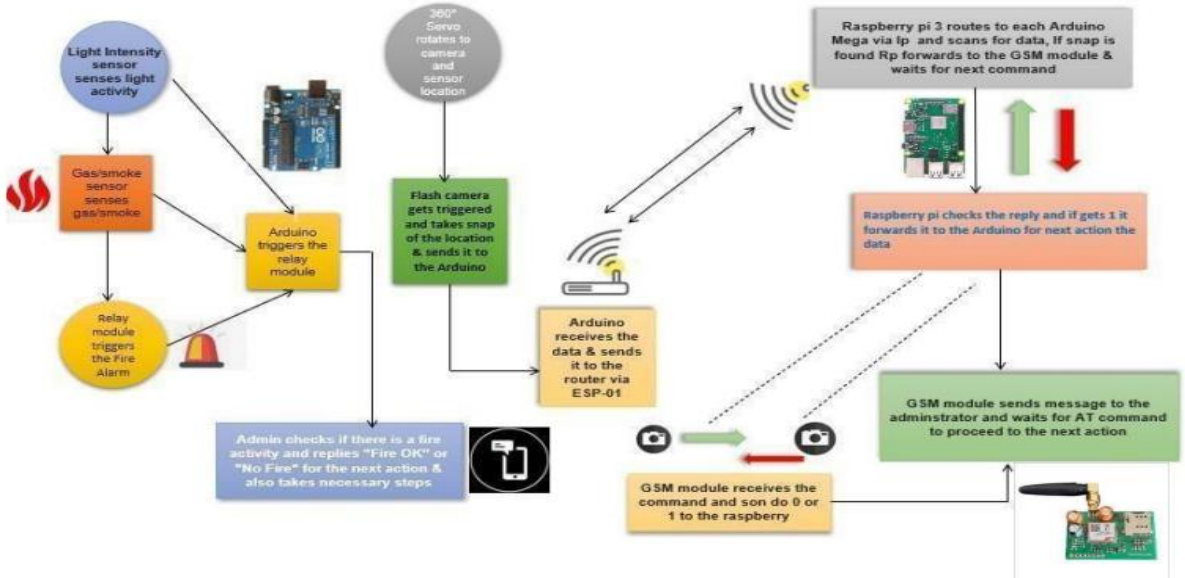
4.2 Non Functional requirement

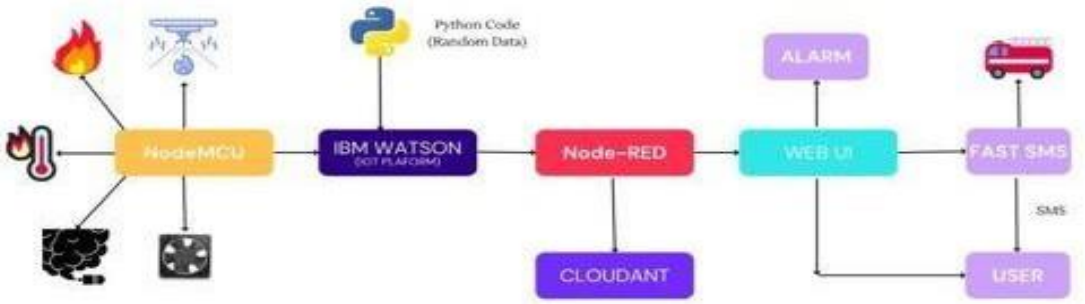
FR No.	Non-Functional Requirement	Description
NFR -1	**Usability**	Usability requirements includes language barriers and localization tasks. Usability can beassessed by Efficiency of use.
NFR -2	**Security**	Access permissions for the particular systeminformation may only be changed by the system's data administrator.
NFR -3	**Reliability**	The database update process must roll back all related updates when any update fails.
NFR -4	**Performance**	The front-page load time must be no more than2 seconds for users that access the website using an VoLTE mobile connection.

5. PROJECT DESIGN
5.1. Data Flow Diagrams



5.2 Solution & Technical Architecture





6.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Release
Sensing	Sensing	USN-1	Sensing the environment using the sensors.	3	High	Sprint-1
	Operating	USN-2	Turning on the exhaust fan as well as the fire sprinkler system in cause of fire and gas leakage.	3	Medium	Sprint-1
Sensor Data	Sending collected data to the IBM Watson platform	USN-3	Sending the data of the Sensors to the IBM Watson.	3	High	Sprint-2
	Registration	USN-4	Entering my email and password to verify authentication process.	3	High	Sprint-2
	Storing of sensor data	USN-5	Storing in Cloudant database.	2	Medium	Sprint-3
	Node red	USN-6	Sending the data from the IBM Watson to the Node red.	3	High	Sprint-3
Web User	Web UI	USN-7	Monitors the situation of the environment which displays sensor information.	1	Low	Sprint-3
Notification	Fast SMS Service	USN-8	Use Fast SMS to Send alert message once the parameters like temperature, flame and gas sensor readings goes beyond the threshold value.	3	High	Sprint-4
Extinguish	Turn ON/OFF the actuators	USN-9	User can turn off the Exhaust fan as well as the sprinkler system If need in that Situation.	2	Medium	Sprint-4
	Testing	USN-10	Testing of project and Final Deliverables.	1	Low	Sprint-4

6. PROJECT PLANNING & SCHEDULING
6.1 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Sensing	USN-1	Use the sensors to sense the surroundings.	3	High	Giritharan Aravind Chidambaranathan Arumugaviswanathan
	Operating	USN-2	Activating the fire sprinkler system and exhaust fan in case of a fire	3	Medium	Giritharan Aravind Chidambaranathan Arumugaviswanathan
Sprint-2	Sending collected data tothe IBM Watsonplatform	USN-3	Sending IBM Watson the data from the sensors.	3	High	Giritharan Aravind Chidambaranathan Arumugaviswana than

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
	Node red	USN-4	Data transmission from IBM Watson to Node Red.	3	High	Giritharan Aravind Chidambaranathan Arumugaviswanathan
Sprint-3	Storing of sensor data	USN-5	Keeping data in a Cloudant database.	2	Medium	Giritharan Aravind Chidambaranathan Arumugaviswanathan
	Registration	USN-6	My email and password are being entered to confirm the authentication process.	1	Medium	Giritharan Aravind Chidambaranathan Arumugaviswanathan
	Web UI	USN-7	Keeps track of environmental conditions and presents sensor data.	3	High	Giritharan Aravind Chidambaranathan Arumugaviswa nathan
Sprint-4	Fast SMS Service	USN-8	When parameters like temperature, flame, and gas sensor readings exceed the threshold value, use Fast SMS to send an alarm message.	3	High	Giritharan Aravind Chidambaranathan Arumugaviswanathan
	Turn ON/OFF the actuators	USN-9	In that case, the user has the option to turn off both the sprinkler system and the exhaust fan.	2	Medium	Giritharan Aravind Chidambaranathan Arumugaviswanathan

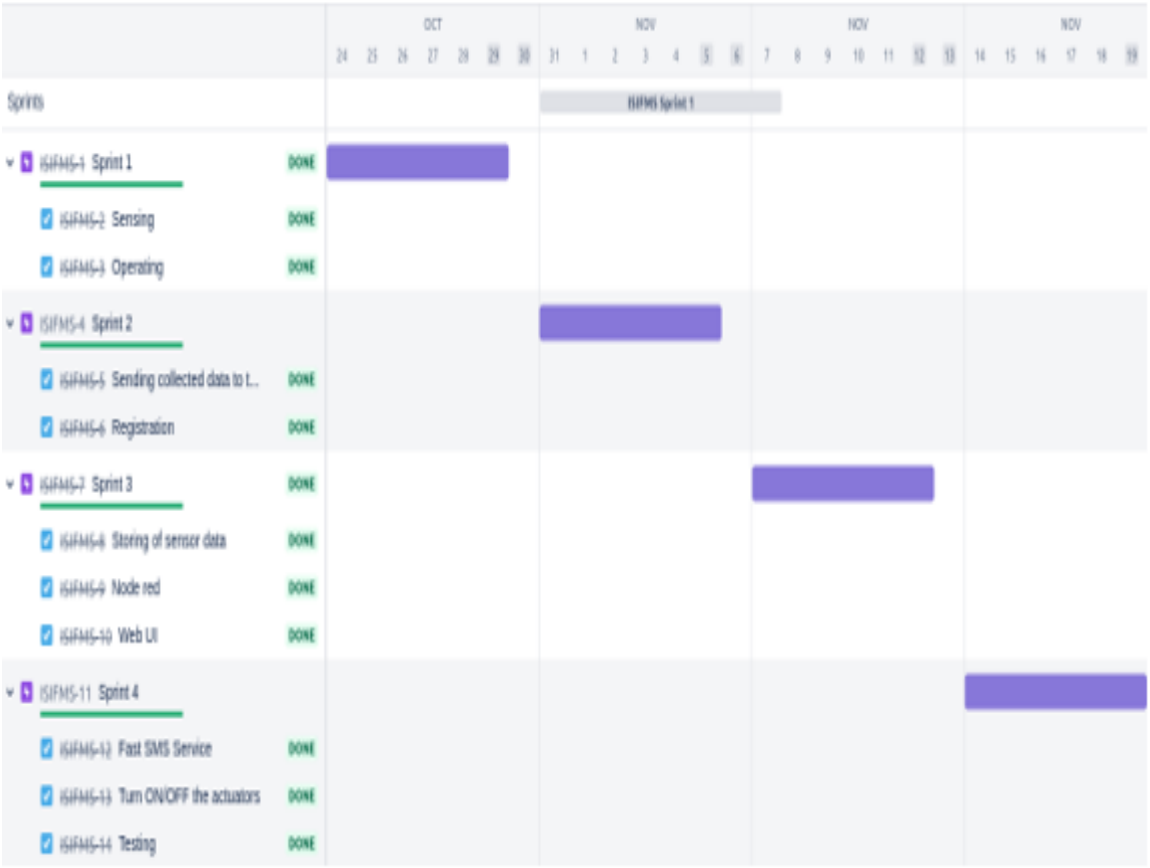
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
	Testing	USN-10	Project and final deliverables testing.	1	Low	Giritharan Aravind Chidambaranathan Arumugaviswanathan

6.2 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	6	6 Days	13 NOV 2022	19 NOV 2022	6	19 Oct 2022
Sprint-2	6	6 Days	13 NOV 2022	19 NOV 2022	6	19 NOV 2022
Sprint-3	6	6 Days	13 NOV 2022	19 NOV 2022	6	19 NOV 2022

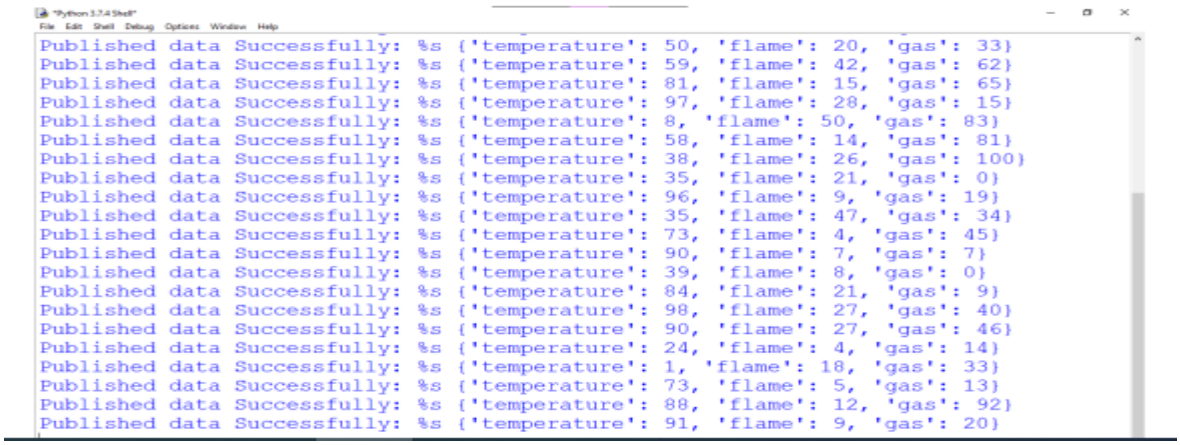
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-4	6	6 Days	13 NOV 2022	19 NOV 2022	6	19 Nov 2022

6.3 Reports from JIRA



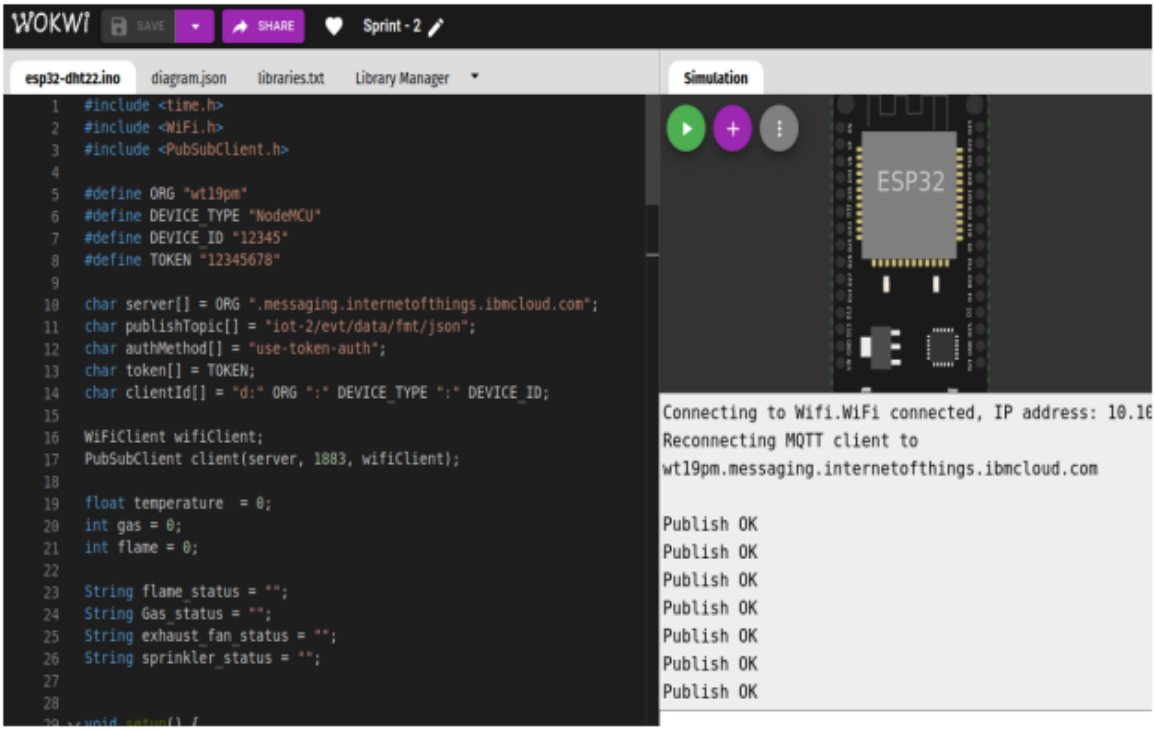
Sprint 1

```
File  Edit  Format  Run  Options  Window  Help
#IBM Watson IOT Platform

#pip install wiotp-sdk import wiotp.sdk.device import time



import random
myConfig = {
        "identity": {
            "orgId": " gltlbd",
            "typeId": "ggg",
            "deviceId":"123"
},
"auth": {
"token": "12345678"

}
}

def myCommandCallback(cmd):
print("Message received from IBM IoT Platform: %s" % cmd.data['command'])
m=cmd.data['command']
client = wiotp.sdk.device.DeviceClient(config=myConfig, logHandlers=None)

client.connect() while True:
temp=random.randint(-5,100) #hum=random.randint(0,100) flame=random.randint(0,50) gas=random.randint(0,100) myData={'temperature':temp,'flame':flame,'gas':g
client.publishEvent(eventId="status", msgFormat="json", data=myData, qos=0, onPublish=None)
print("Published data Successfully: %s", myData) client.commandCallback = myCommandCallback time.sleep(4)
client.disconnect()
```



```
"Python 3.7.4 Shell"
File  Edit  Shell  Debug  Options  Window  Help
Published data Successfully: %s {'temperature': 50, 'flame': 20, 'gas': 33}
Published data Successfully: %s {'temperature': 59, 'flame': 42, 'gas': 62}
Published data Successfully: %s {'temperature': 81, 'flame': 15, 'gas': 65}
Published data Successfully: %s {'temperature': 97, 'flame': 28, 'gas': 15}
Published data Successfully: %s {'temperature': 8, 'flame': 50, 'gas': 83}
Published data Successfully: %s {'temperature': 58, 'flame': 14, 'gas': 81}
Published data Successfully: %s {'temperature': 38, 'flame': 26, 'gas': 100}
Published data Successfully: %s {'temperature': 35, 'flame': 21, 'gas': 0}
Published data Successfully: %s {'temperature': 96, 'flame': 9, 'gas': 19}
Published data Successfully: %s {'temperature': 35, 'flame': 47, 'gas': 34}
Published data Successfully: %s {'temperature': 73, 'flame': 4, 'gas': 45}
Published data Successfully: %s {'temperature': 90, 'flame': 7, 'gas': 7}
Published data Successfully: %s {'temperature': 39, 'flame': 8, 'gas': 0}
Published data Successfully: %s {'temperature': 84, 'flame': 21, 'gas': 9}
Published data Successfully: %s {'temperature': 98, 'flame': 27, 'gas': 40}
Published data Successfully: %s {'temperature': 90, 'flame': 27, 'gas': 46}
Published data Successfully: %s {'temperature': 24, 'flame': 4, 'gas': 14}
Published data Successfully: %s {'temperature': 1, 'flame': 18, 'gas': 33}
Published data Successfully: %s {'temperature': 73, 'flame': 5, 'gas': 13}
Published data Successfully: %s {'temperature': 88, 'flame': 12, 'gas': 92}
Published data Successfully: %s {'temperature': 91, 'flame': 9, 'gas': 20}
```

Sprint 2

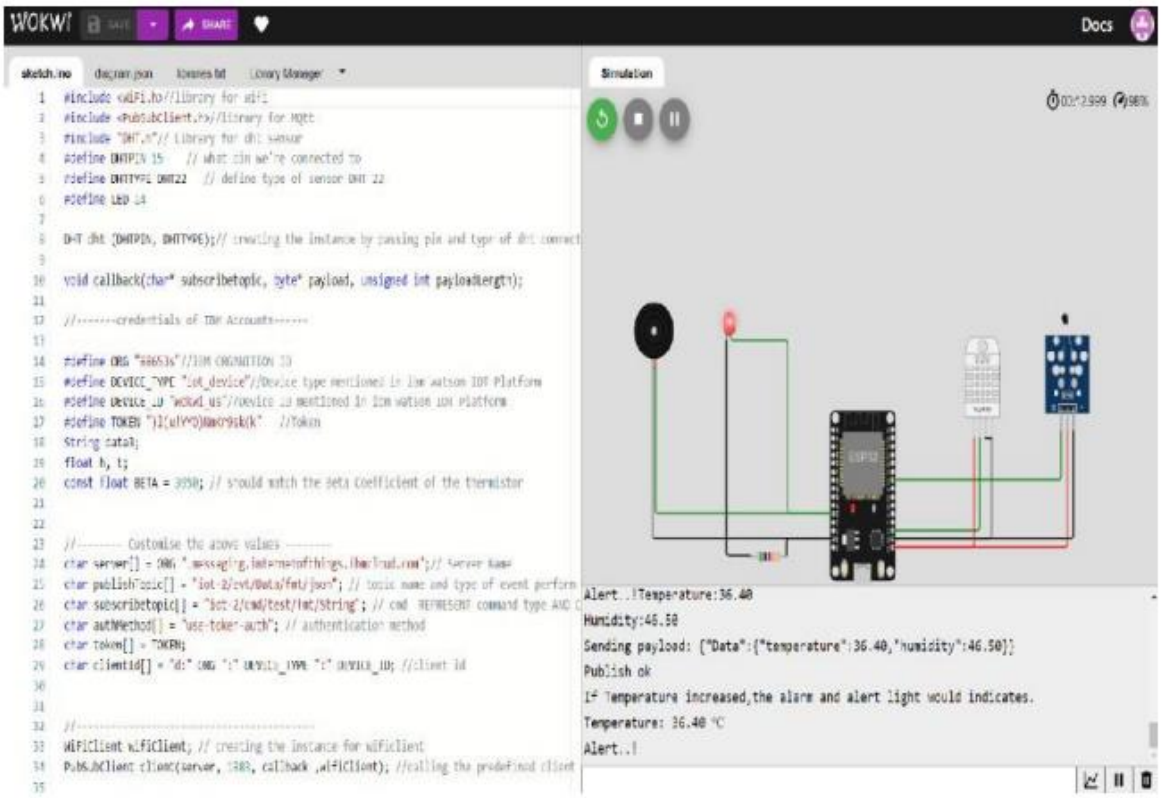


```
WOKWI  SAVE  SHARE  Sprint - 2
esp32-dht22.ino  diagram.json  libraries.txt  Library Manager
Simulation
```

```
#include <time.h>
#include <WiFi.h>
#include <PubSubClient.h>

#define ORG "wt19pm"
#define DEVICE_TYPE "NodeMCU"
#define DEVICE_ID "12345"
#define TOKEN "12345678"

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";
char publishTopic[] = "iot-2/evt/data/fmt/json";
char authMethod[] = "use-token-auth";
char token[] = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;

WiFiClient wifiClient;
PubSubClient client(server, 1883, wifiClient);

float temperature  = 0;
int gas = 0;
int flame = 0;

String flame_status = "";
String Gas_status = "";
String exhaust_fan_status = "";
String sprinkler_status = "";
```

```
ESP32

Connecting to Wifi.WiFi connected, IP address: 10.10
Reconnecting MQTT client to
wt19pm.messaging.internetofthings.ibmcloud.com

Publish OK
Publish OK
Publish OK
Publish OK
Publish OK
Publish OK
Publish OK
```

Sprint 3



7.CODING AND SOLUTIONS

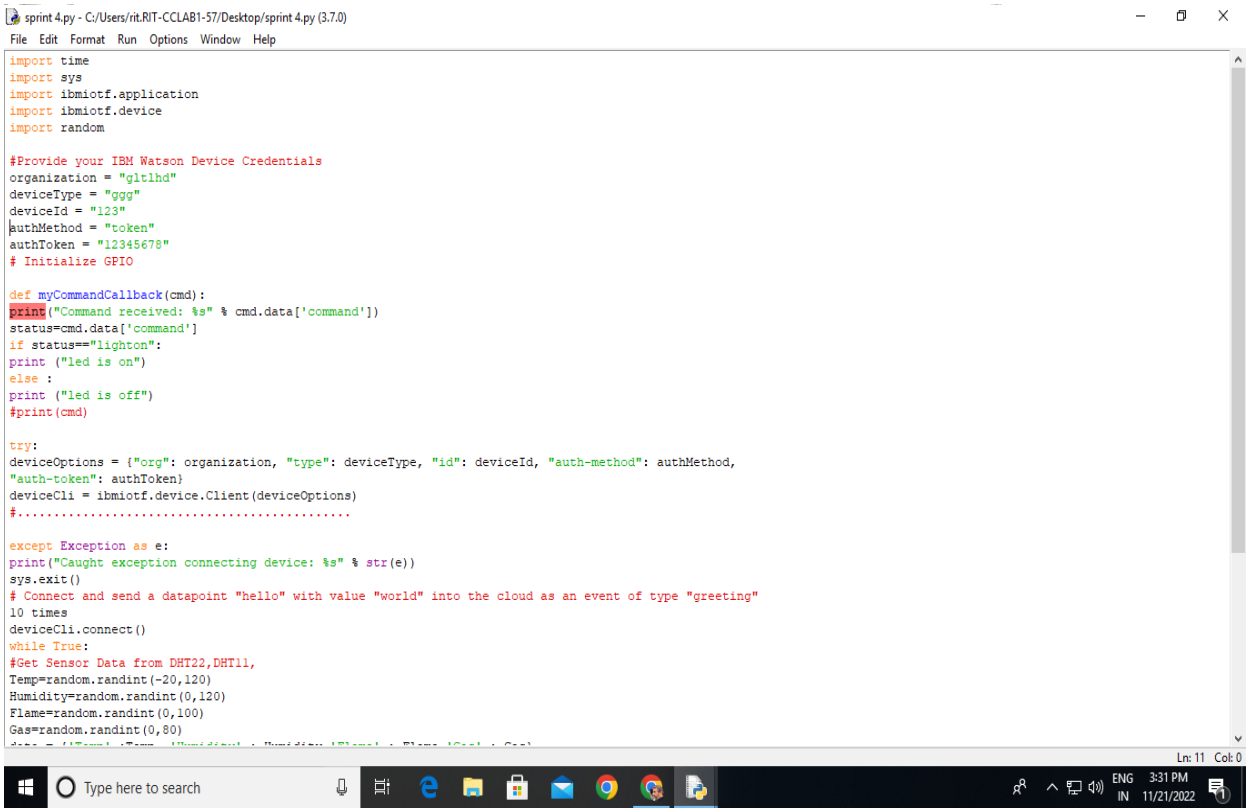
7.1 Features

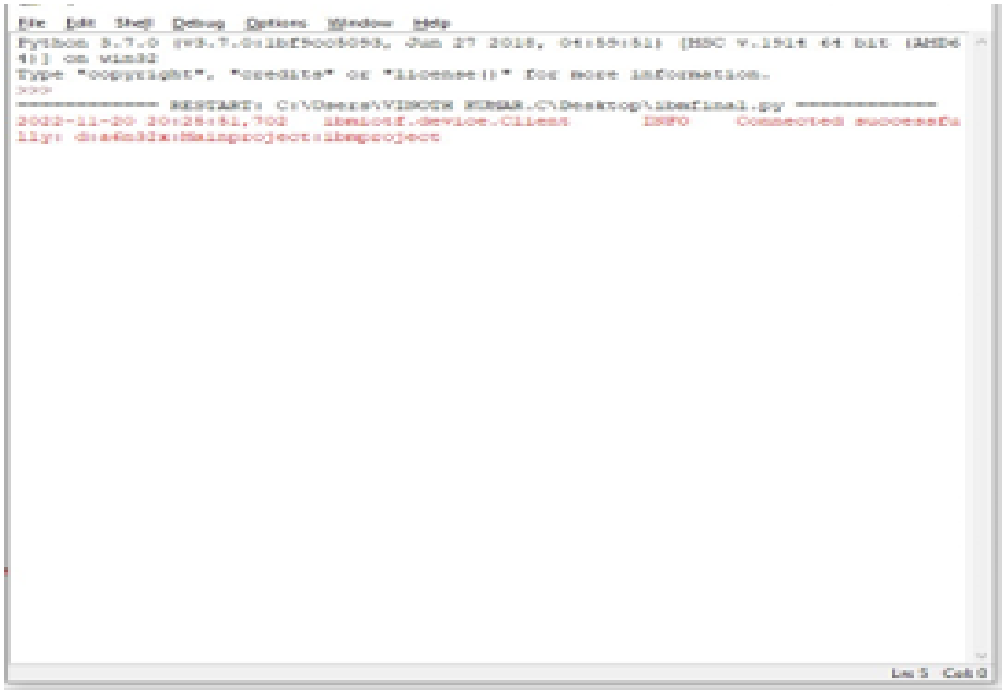
```
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random

#Provide your IBM Watson Device Credentials
organization = "gltlhd"
deviceType = "ggg"
deviceId = "123"
authMethod = "token"
authToken = "12345678"
# Initialize GPIO
```

```
def myCommandCallback(cmd):
print("Command received: %s" % cmd.data['command'])
status=cmd.data['command']
if status=="lighton":
print ("led is on")
else :
print ("led is off")
#print(cmd)

try:
deviceOptions = {"org": organization, "type": deviceType, "id": deviceId,
"auth-method": authMethod,
"auth-token": authToken}
deviceCli = ibmiotf.device.Client(deviceOptions)
#.....

except Exception as e:
print("Caught exception connecting device: %s" % str(e))
sys.exit()
# Connect and send a datapoint "hello" with value "world" into the cloud as an
event of type "greeting"
10 times
deviceCli.connect()
while True:
#Get Sensor Data from DHT22,DHT11,
Temp=random.randint(-20,120)
Humidity=random.randint(0,120)
Flame=random.randint(0,100)
Gas=random.randint(0,80)
data = {'Temp' :Temp ,'Humidity' : Humidity,'Flame' : Flame,'Gas' : Gas}

def myOnPublishCallback():
if Flame > 100:
data = {'Flame' : Flame}

print ("Temperature =%s c" % Temp ,"Humidity =%s u" % Humidity,"Flame
=%s ir" % Flame ,"Gas
=%s ppm" % Gas )
success = deviceCli.publishEvent("IoTSensor", "json", data, qos=0,
on_publish=myOnPublishCallback)
```

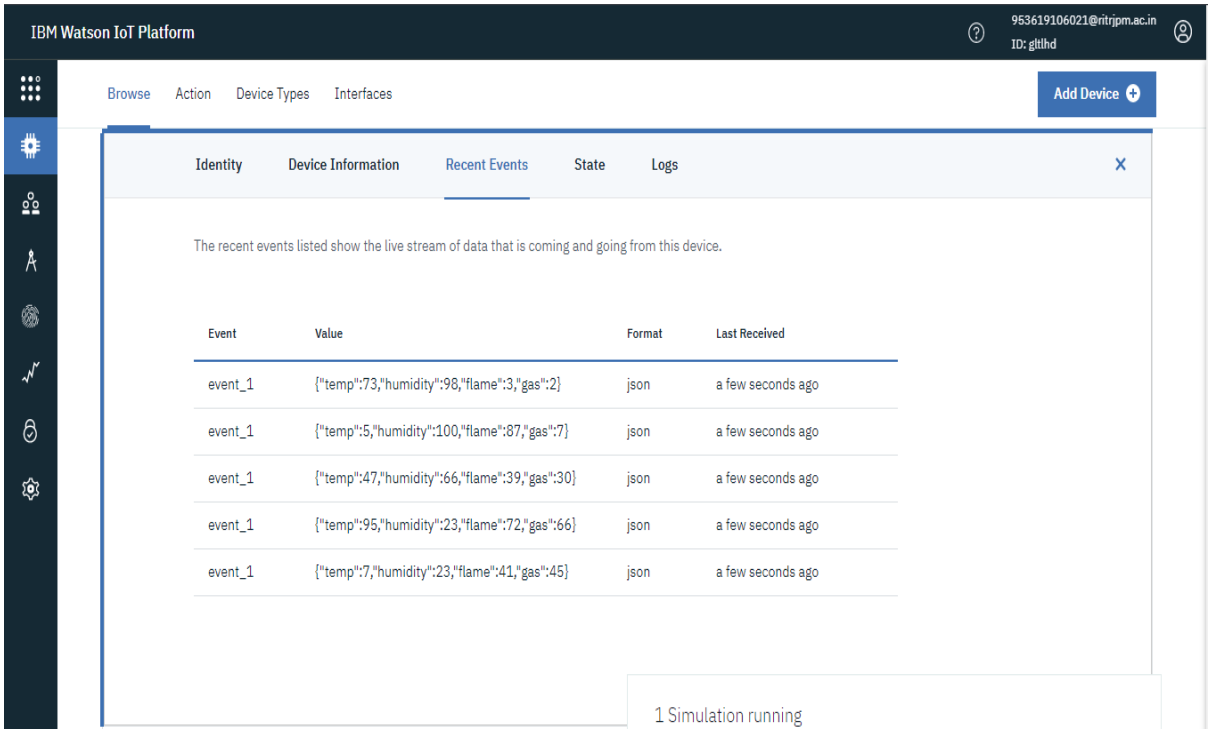
if not success:
print("Not connected to IoTF")
time.sleep(1)
deviceCli.commandCallback = myCommandCallback
# Disconnect the device and application from the cloud
deviceCli.disconnect()

PYTHON CODE OUTPUT

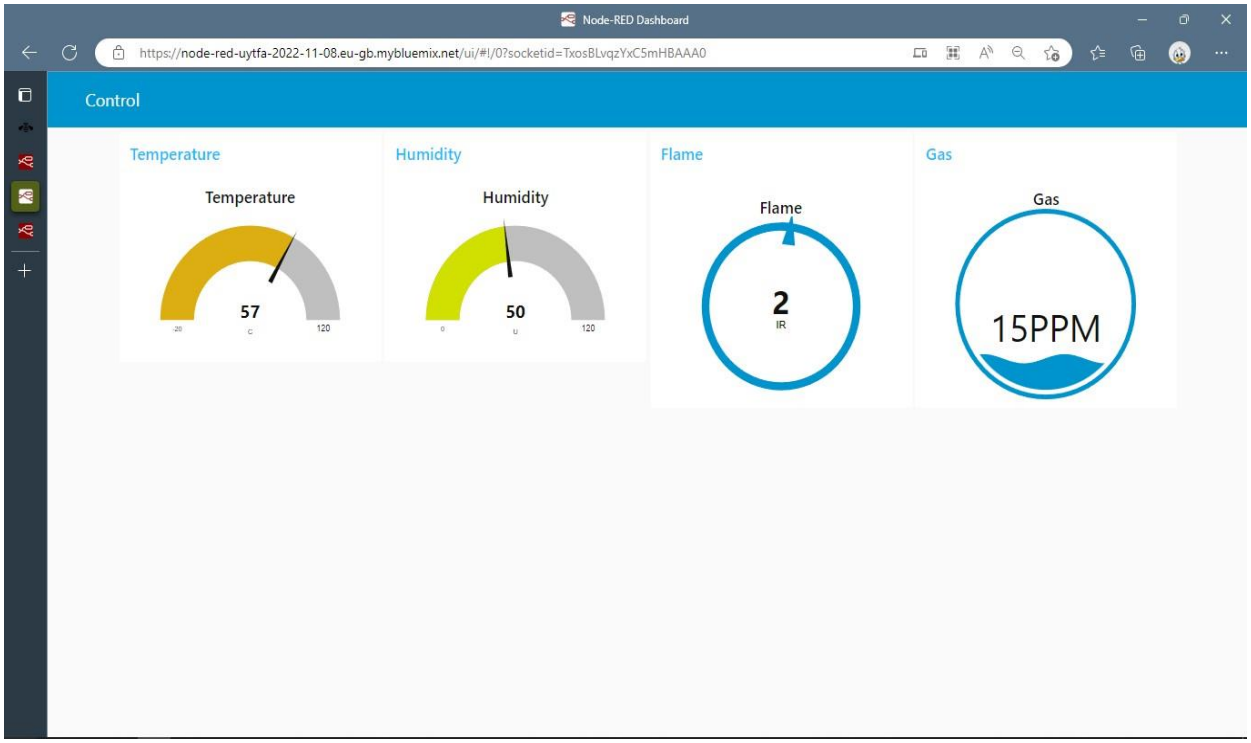




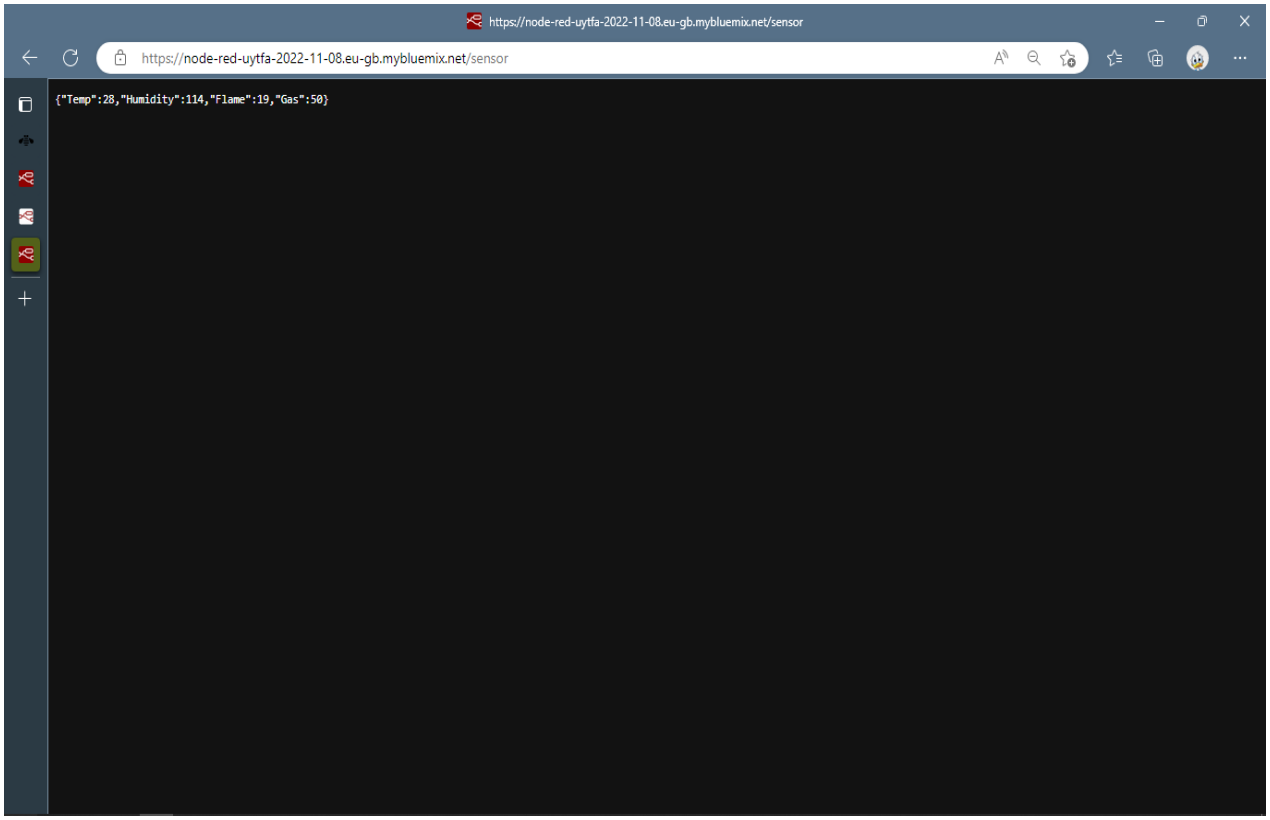
IBM WATSON OUTPUT



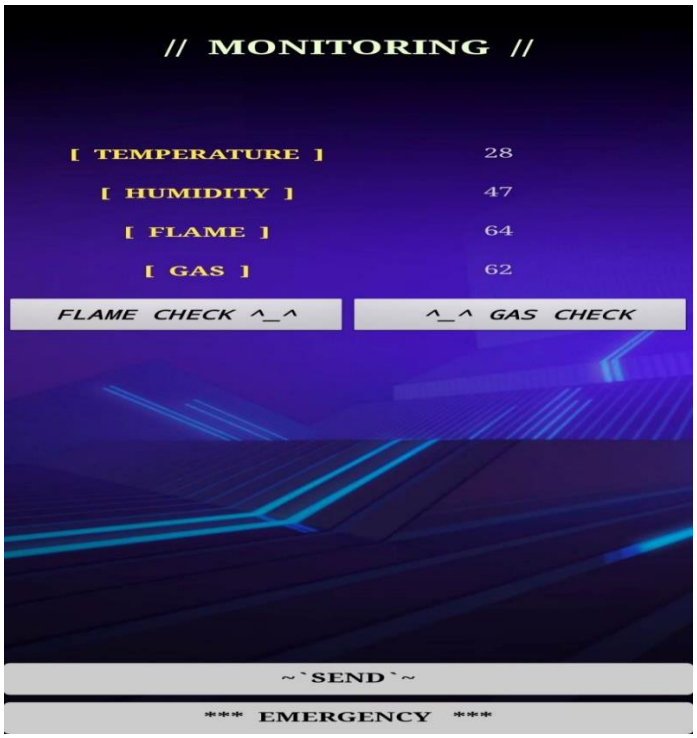
NODERED UI OUTPUT

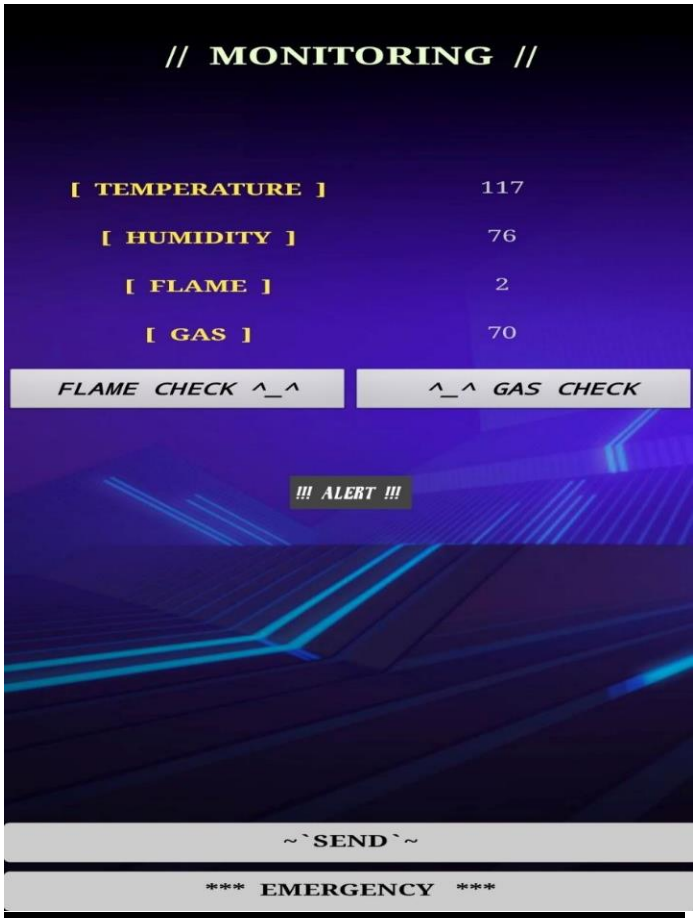


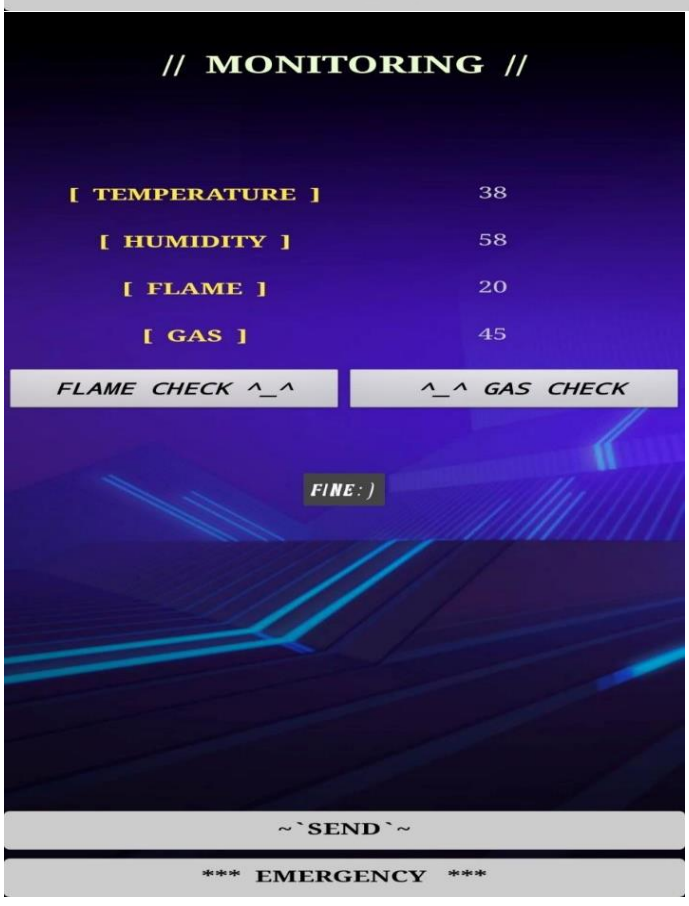
NODE RED SENSOR READING



APP OUTPUT:




// MONITORING //
[TEMPERATURE] 117
[HUMIDITY] 76
[FLAME] 2
[GAS] 70
FLAME CHECK ^_^
^_^ GAS CHECK
!!! ALERT !!!
~`SEND`~
*** EMERGENCY ***


// MONITORING //
[TEMPERATURE] 38
[HUMIDITY] 58
[FLAME] 20
[GAS] 45
FLAME CHECK ^_^
^_^ GAS CHECK
FINE:)
~`SEND`~
*** EMERGENCY ***

8.Testing

8.1. Test cases

Date	15-Nov-22
Team ID	PNT2022TMID17652
Project Name	Project - Industry - Specific Intelligent
Maximum Marks	4 marks

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Date	Expected Result	Actual Result	Status	Executed By	TC for Automation(Y/N)
TC_OO1	Functional	IBM cloud	Create the IBM Cloud services which are being used in this project.	IBM Cloud Login ID & Password	1.Go to IBM Cloud signup page 2.Enter e-mail id and other credentials 3.Enter a password	https://cloud.ibm.com/login	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO2	Functional	IBM Cloud	Configure the IBM Cloud services which are being used in completing this project.	IBM Cloud Login ID & Password	1.Go to Cloud login 2.Enter user ID & Password 3.Verify login by the popup display	https://cloud.ibm.com/login	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO3	Functional	IBM Watson IoT Platform	IBM Watson IoT platform acts as the mediator to connect the web application to IoT devices, so create the IBM Watson IoT platform.	IBM Watson IoT Platform Login ID & Password	1.Login to IBM Cloud 2.Click Catalog 3.Search IoT and click create 4.Go to resource list and search Internet of Things platform 5.Press Launch and click Sign in IBM Watson Platform	https://w1nouz.internetofthings.ibmcloud.com/dashboard/devices/browse	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO4	Functional	IBM Watson	In order to connect the IoT device to the IBM cloud, create a device in the IBM Watson IoT platform and get the device credentials.	IBM Watson IoT Platform Login ID & Password	1.Login to IBM Watson Platform 2. Click Add Device 3.Enter the details and click Finish. Create Device ID & Device type 4.Turn on Device Simulator and click simulation running. Enter the values of temperature,pH & turbidity level 5.Click Send & Save. Verify the displayed result of the levels	Temperature, pH and turbidity sensor values are generated randomly in simulation	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO5	Functional	IBM Cloud(Node Red)	Configure the connection security and create API keys that are used in the Node-RED service for accessing the IBM IoT Platform.	Node Red Installation	1.Install node red and open node red in command prompt 2.Select IBM input in IoT	https://cloud.ibm.com/developer/appservice/create-app?starterKit=59c5d5bd-4d31-9611-897a-f94eea80dc9f&defaultLanguage=undefined	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO6	Functional	Node Red	Create a Node-RED service.	Node Red Installation	1.Select IBM IoT input in Node. In IBM IoT Watson Platform, go to apps and click on generate API keys 2.Copy & paste generated API key and token in the IBM IoT input. After entering all details, click the done button. 3.Add debug to the IBM IoT and rename as Msg payload and click on done. Click gauge from the dashboard and fill the details & add functions to the gauge. Check the generated values from the debug message. 4.Edit function node, connect them, add another gauge and functions, name them as "Temperature", "Gas" & "Humidity" 5.Finally add light ON/OFF buttons to the IBM IoT and debug. Verify the output from NODE RED using Local host link	Values of sensors and button for light ON/OFF is displayed	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO7	Functional	Python 3.7.0	Develop a python script to publish random sensor data such as temperature, humidity level and Gas level to the IBM IoT platform	Python 3.7.0(64 bit) installation	1.Download and install Python 3.7.0 2.Develop python code	https://www.python.org/downloads/release/python-370/	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO8	Functional	Python 3.7.0	After developing python code, commands are received just print the statements which represent the control of the devices.	Python 3.7.0(64 bit) installation	1.Downlinstall Python 3.7.0 2.After python code	Get the output from the code	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_OO9	Functional	M Cloudant D	Publish Data to The IBM Cloud	IBM Cloud Login ID & Password	1.Run the python code 2.Verify the displayed output	Publishment of python code	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
Sensor	Functional	ESP32 Controller	Sensor data is taken and parsed as JSON	Circuit Connections	1.Open the simulator in wokwi and simulate the device	Random value generated	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_O11	Storage	IBM Cloudant DB	Configure the Node-RED flow to receive data from the IBM IoT platform and also use Cloudant DB nodes to store the received sensor data in the cloudant DB		1.Go to IBM cloud, search Cloudant in Catalog, Add new dashboard, go to Node Red 2.Connect to cloudant and verify the results	Cloudant is connected by NODE RED	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N
TC_O12	API	SMS API	The SMS is sent when there is Fire Alert.	The node red should be configured to send a post request	1. Simulate the fire in the simulator 2. or click the sent alert button in	"Fire alert at Industries Hurry" and the trigger inputs	Results verified	No	Pass	Manikandan B , Kannaka Subbu Laakshmi B R , Bhavithra G , Arunagirinathan S	N

UAT

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	11	5	2	3	21
Duplicate	1	0	3	0	4
External	4	5	0	1	10
Fixed	10	2	3	20	35
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	26	17	12	26	81

9. RESULTS
9.1. Performance metrics

1. Hours worked : 48 hours
2. Efficiency of the product : 100%
3. Quality of the product : 100%

10. ADVANTAGES
It reduces false alarms.
It has a minimal installation cost, and continuously scans the area.
It enhances security in workplaces and businesses.

DISADVANTAGES
Large-scale industries cannot use this system.
If the control panel is damaged, it needs to be replaced.

11. CONCLUSION
This technology notifies the authorities of the situation at the appropriate moment, decreasing the number of false warnings. Since the technology is affordable, small scale industries can readily implement it.

12. FUTURE SCOPE
With the addition artificial intelligent technology, Fire management system can be made automated. With the use of PIR(Passive Infrared Sensor) the count of human can be detected in that area and prioritize it, which helps in human life saving.

13. APPENDIX
13.1. Source code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <time.h>
#include "DHTesp.h"
#define temp_pin 15
void callback(char* subscribetopic,byte* payload, unsigned int payloadLength);
#define ORG " gltlhd "
#define DEVICE_TYPE "ggg"
#define DEVICE_ID "123"
#define TOKEN "12345678"
String data3;
```

```
char server[]= ORG ".messaging.internetofthings.ibmcloud.com";
char publishTopic[]="iot-2/evt/Data/fmt/json";
char subscribeTopic[]="iot-2/cmd/test/fmt/String";
char authMethod[]="use-token-auth";
char token[]=TOKEN;
char clientID[]="d:"ORG":"DEVICE_TYPE":"DEVICE_ID;
WiFiClient wifiClient;
PubSubClient client(server,1883,callback,wifiClient);

const int DHT_PIN = 15;

DHTesp dhtSensor;


bool exhaust_fan_on = false;
bool sprinkler_on = false;

float temperature  = 0;
int gas = 0;
int flame = 0;

String flame_status = "";
String accident_status = "";
String sprinkler_status = "";

void setup() {
   Serial.begin(99900);

wificonnect();
mqttconnect();

  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);
}


void loop() {
  srand(time(0));

    //initial variable

    temperature = random(-20,125);
    gas = random(0,1000);
    int flamereading = random(200,1024);
```

```
    flame = map(flamereading,0,1024,0,2);

  TempAndHumidity  data = dhtSensor.getTempAndHumidity();

Serial.println("Temperature: "+ String(data.temperature, 2) + "°C");
Serial.println("Humidity: " + String(data.humidity, 1) + "%");
  Serial.println("---");
  delay(1000);
if(data.temperature<38){PublishData1(data.temperature);
        flame_status = "No Fire";
        Serial.println("Flame Status : "+flame_status);
    }
    else{ PublishData2(data.temperature);
        flame_status = "Fire is Detected";
        Serial.println("Flame Status : "+flame_status);

    }
    if(data.humidity<30){PublishData3(data.humidity);
        Serial.println("Gas Status : Gas leakage Detected");
    }
    else{PublishData4(data.humidity);
        exhaust_fan_on = false;
        Serial.println("Gas Status : No Gas leakage Detected");
    }

    //send the sprinkler status
    if(data.temperature<38){
        sprinkler_status = " not working";
        Serial.println("Sprinkler Status : "+sprinkler_status);
    }
    else{
        sprinkler_status = " working";
        Serial.println("Sprinkler Status : "+sprinkler_status);
    }

    //toggle the fan according to gas

    if(data.humidity<30){
        exhaust_fan_on = true;
        Serial.println("Exhaust fan Status : Working");
    }
    else{
        exhaust_fan_on = false;
```

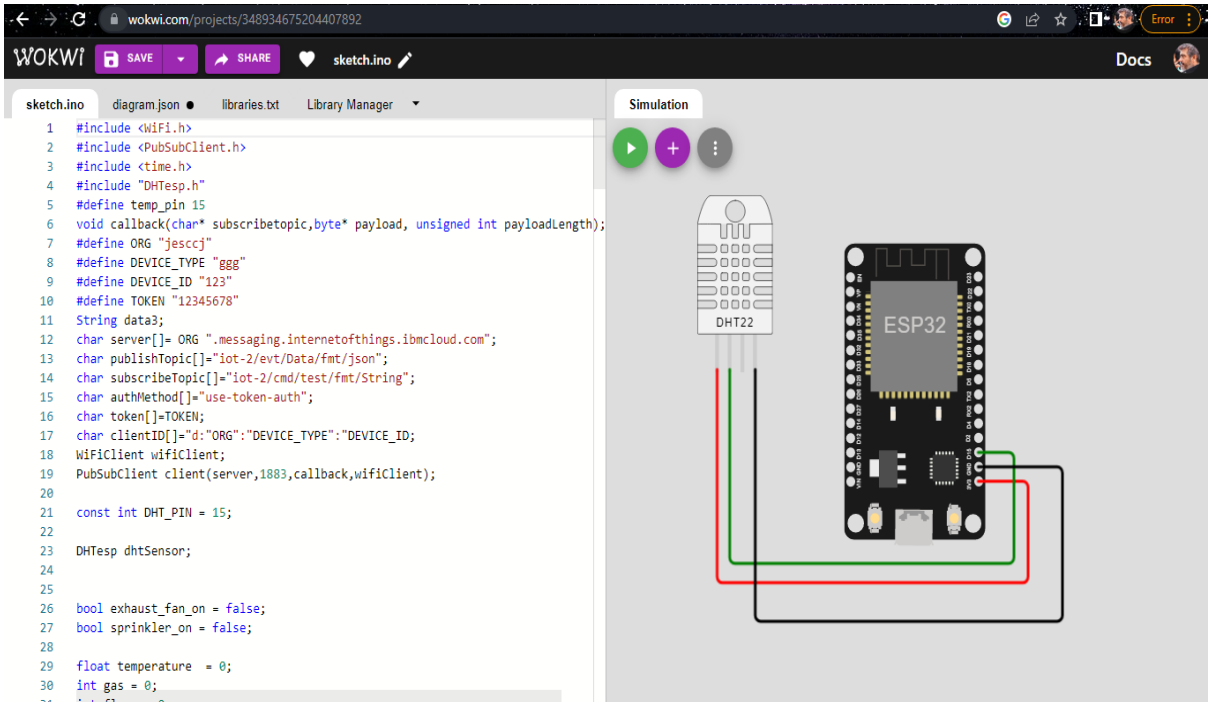
```
      Serial.println("Exhaust fan Status : Not Working");
    }

    Serial.println("");
    Serial.println("");
    Serial.println(" ------------------------**********----------------------");
    Serial.println("");
    Serial.println("");
```

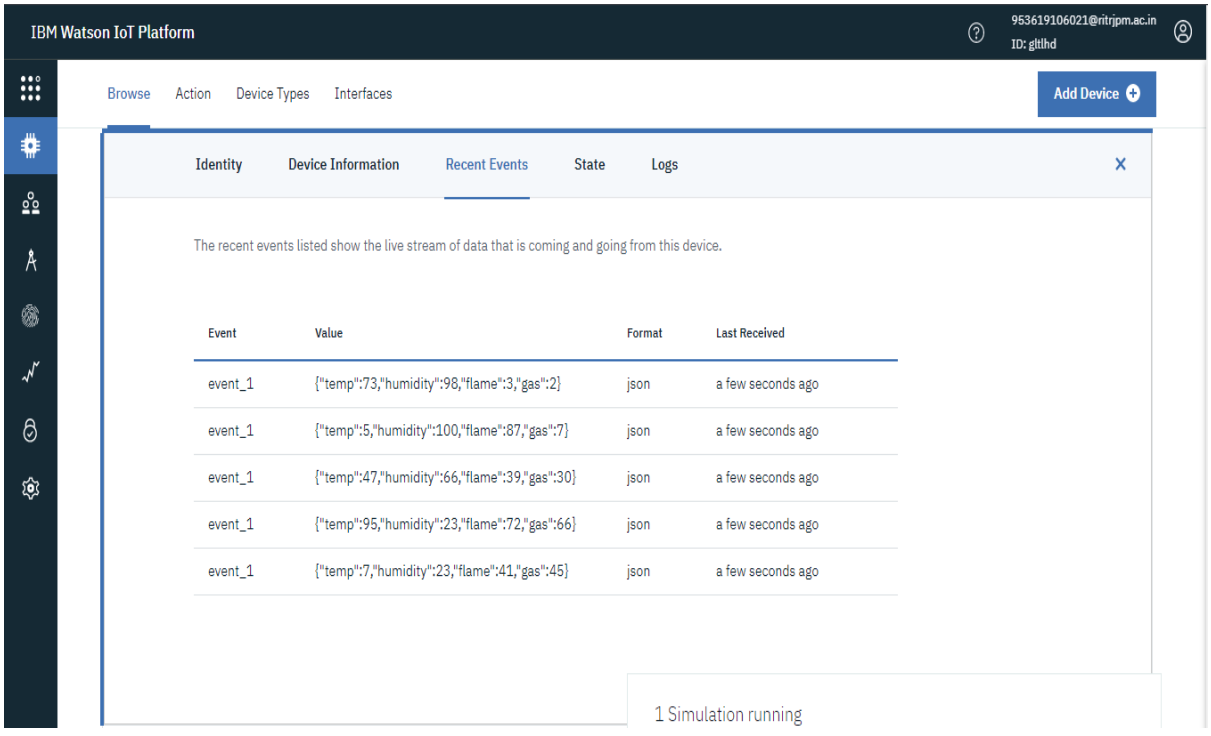
```
delay(1000);
if(!client.loop()){
mqttconnect();
}
}void PublishData1(float temp){
  mqttconnect();
  String payload = "{\"temp\":";
payload += temperature;
payload += ",\"nrml!\":""\"temperature less than 38\"";
payload += "}";
Serial.print("Sending payload: ");
Serial.println(payload);


  if(client.publish(publishTopic,(char*)payload.c_str())){
    Serial.println("publish ok");
  } else{
    Serial.println("publish failed");
  }
}
void PublishData2(float temperature){
  mqttconnect();
  String payload = "{\"temp\":";
payload += temperature;
payload += ",\"ALERT!!\":""\"temperature greater than 38\"";
payload += "}";
Serial.print("Sending payload: ");
Serial.println(payload);
  if(client.publish(publishTopic,(char*)payload.c_str())){
    Serial.println("publish ok");
```

```
  } else{
    Serial.println("publish failed");
  }
}
void PublishData3(float humidity){
  mqttconnect();
  String payload = "{\"hum\":";
payload += humidity;
payload += ",\"ALERT!!\":""\"humidity less than 30\"";
payload += "}";
Serial.print("Sending payload: ");
Serial.println(payload);
  if(client.publish(publishTopic,(char*)payload.c_str())){
    Serial.println("publish ok");
  } else{
    Serial.println("publish failed");
  }
}
void PublishData4(float humidity){
  mqttconnect();
  String payload = "{\"hum\":";
payload += humidity;
payload += ",\"nrml!!\":""\"humidity greater than 30\"";
payload += "}";
Serial.print("Sending payload: ");
Serial.println(payload);


  if(client.publish(publishTopic,(char*)payload.c_str())){
    Serial.println("publish ok");
  } else{
    Serial.println("publish failed");
  }
}
void mqttconnect(){
  if(!client.connected()){
    Serial.print("Reconnecting to");
    Serial.println(server);
    while(!!!client.connect(clientID, authMethod, token)){
      Serial.print(".");
      delay(500);
    }
    initManagedDevice();
```

```
    Serial.println();
  }
}

void wificonnect(){
  Serial.println();
  Serial.print("Connecting to");

  WiFi.begin("Wokwi-GUEST","",6);
  while(WiFi.status()!=WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WIFI CONNECTED");
  Serial.println("IP address:");
  Serial.println(WiFi.localIP());
}

void initManagedDevice(){
  if(client.subscribe(subscribeTopic)){
    Serial.println((subscribeTopic));
    Serial.println("subscribe to cmd ok");
  }else{
    Serial.println("subscribe to cmd failed");
  }
}

void callback(char* subscribeTopic, byte* payload, unsigned int
payloadLength){
  Serial.print("callback invoked for topic:");
  Serial.println(subscribeTopic);
  for(int i=0; i<payloadLength; i++){
    data3 += (char)payload[i];
  }
}
```

OUTPUT





13.2. link

Github link - https://github.com/IBM-EPBL/IBM-Project-38678-1660384396