

GOVERNMENT COLLEGE OF TECHNOLOGY

(An autonomous institution, NAAC accredited, affiliated to Anna University, Chennai)
Thadagam Road, Coimbatore - 641013

INDUSTRY-SPECIFIC INTELLIGENT FIRE MANAGEMENT SYSTEM

DOMAIN – Internet Of Things

IBM – NALAIYATHIRAN

PROJECT REPORT

TEAM MEMBERS:

KAVIYA SHRI M – GCTC1914123

JIGENDHRA GOWRI M – GCTC1914118

KARISHMA B B – GCTC1914119

PREETHI S R – GCTC1914128

TEAM ID: PNT2022TMID06918

FACULTY MENTOR: Mr. Mathivanan

INDUSTRY MENTOR: Santoshi

Title	Page No.
1. INTRODUCTION	3
1.1. Project Overview	3
1.2. Purpose	3
2. LITERATURE SURVEY	4
2.1. References	4
2.2. Proposed Method	5
2.3. Problem Statement Definition	6
3. IDEATION AND PROPOSED SOLUTION	6
3.1. Empathy Map Canvas	6
3.2. Ideation and Brainstorming	7
3.3. Proposed Solution	8
3.4. Problem Solution Fit	9
4. REQUIREMENT ANALYSIS	10
4.1. Functional analysis	10
4.2. Non-Functional analysis	10
5. PROJECT DESIGN	11
5.1. Data Flow diagrams	11
5.2. Solution & Technical Architecture	12
5.3. User Stories	13
5.4. Milestones	13
6. PROJECT PLANNING & SCHEDULING	15
6.1. Technology stack	15
6.2. Sprint Delivery Schedule	16
7. SPRINT DELIVERY	16
7.1. Sprint 1	17
7.2. Sprint 2	23
7.3. Sprint 3	27
7.4. Sprint 4	36
8. RESULTS	39
9. ADVANTAGES	40
10. CONCLUSION	41
11. FUTURE SCOPE	41
12. APPENDIX	41
Source Code	42
GitHub & Project Demo Link	44

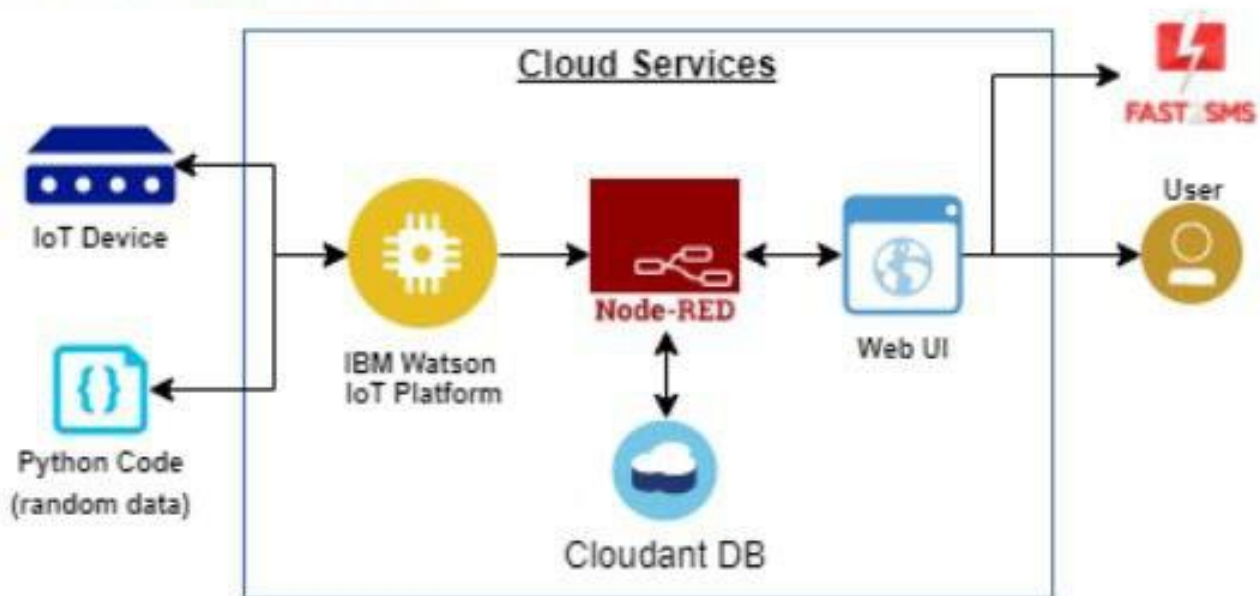
INTRODUCTION:

Fire and smoke kill more people every year than many other accidents. However, the rapid detection of fire and its control can save lives and property damage worth millions. The primary purpose of the fire management system is to provide an early warning of fire so that people can be evacuated and immediate actions can be taken to stop and eliminate its effect as soon as possible.

PROJECT OVERVIEW:

The smart fire management system includes a Gas sensor, Flame sensor and temperature sensors to detect any changes in the environment. Based on the temperature readings and if any Gases are present the exhaust fans are powered ON. If any flame is detected the sprinklers will be switched on automatically. Emergency alerts are notified to the authorities and Fire station.

Technical Architecture:



PURPOSE:

1. An Intelligent solution to detect fire breakdown in industries at early stages so as to alert people working inside and taking necessary counter actions to put off it easily without loss of human lives or material damage.
2. A better alternative from the existing technology, where use usually have alarms notifying about the breakdown.

3. Measures various parameters continuously which can be pushed to cloud and those can be accessed anytime to analyse.
4. Immediate support from nearby fire stations in case of uncontrollable fire which can be simultaneously notified to management authorities of the industry.

LITERATURE SURVEY

REFERENCES:

PAPER TITLE	AUTHOR	OBJECTIVE/OUTCOME
A Survey of Fire Safety Measures for Industry Safety Using IOT	N. Savitha ; S. Malathi 2019	In the system the fire safety practices is going to implement for the fire crackers industry. In that the root cause for the fire is to be analyzed and prevent from the fire before it is triggered. Through this hazardous fire accidents can be avoided and many lives can be saved.
Design of Distributed Factory Fire Alarm Systems	Li Liu ;Yanke C I ; Haosong chen 2020	The Distributed plant fire alarm system can quickly detect the fire and issues an alarm to reduce the damage caused by the fire. The fire alarm system is a control system that integrates signal detection,transmission, processing and control .It mainly complete the basic function of Fire ,smoke and temperature module monitering fire.

A Microcontroller- based Fire Protection System for the Safety of Industries in Bangladesh	Md. Saïam Dept. of Electrical and Electronic Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh 2021	The affected area is also triggered by the fire extinguishing equipment. At the same time, it also notifies the manager and the nearby fire station via SMS. This paper presents a simulation and practical arrangement of the system to demonstrate how it can be implemented as a fire prevention equipment.
Safety Robot for Flammable Gas and Fire Detection using Multisensor Technology	Sandeep Prabhakaran ; Mathan N	In case of fire accidents, the robot alerts the workstation and sends a mail to the firefighting department with the location read from the GPS module. As the robot works as an autonomous system, it does not need to be controlled remotely. Hence this robot is based on the line following mechanism, it is quite easy to install and can cover a large area efficiently.
Computer Vision Based Industrial and Forest Fire Detection Using Support Vector Machine (SVM)	Md. Abdur Rahman ; Sayed Ta++++nimun Hasan ; Mohammed Abdul Kader 2022	The proposed strategy works on a very large dataset of fire videos that have been collected both in real-life situations and from the internet. This SVM pipeline model shows the maximum accuracy is 93.33%. The system can fulfill the precision and detect faster real-time fire detection. It's forest and industrial application will aid in the early detection of fires, as well as emergency management, and so immensely contribute to loss prevention.

PROPOSED METHOD:

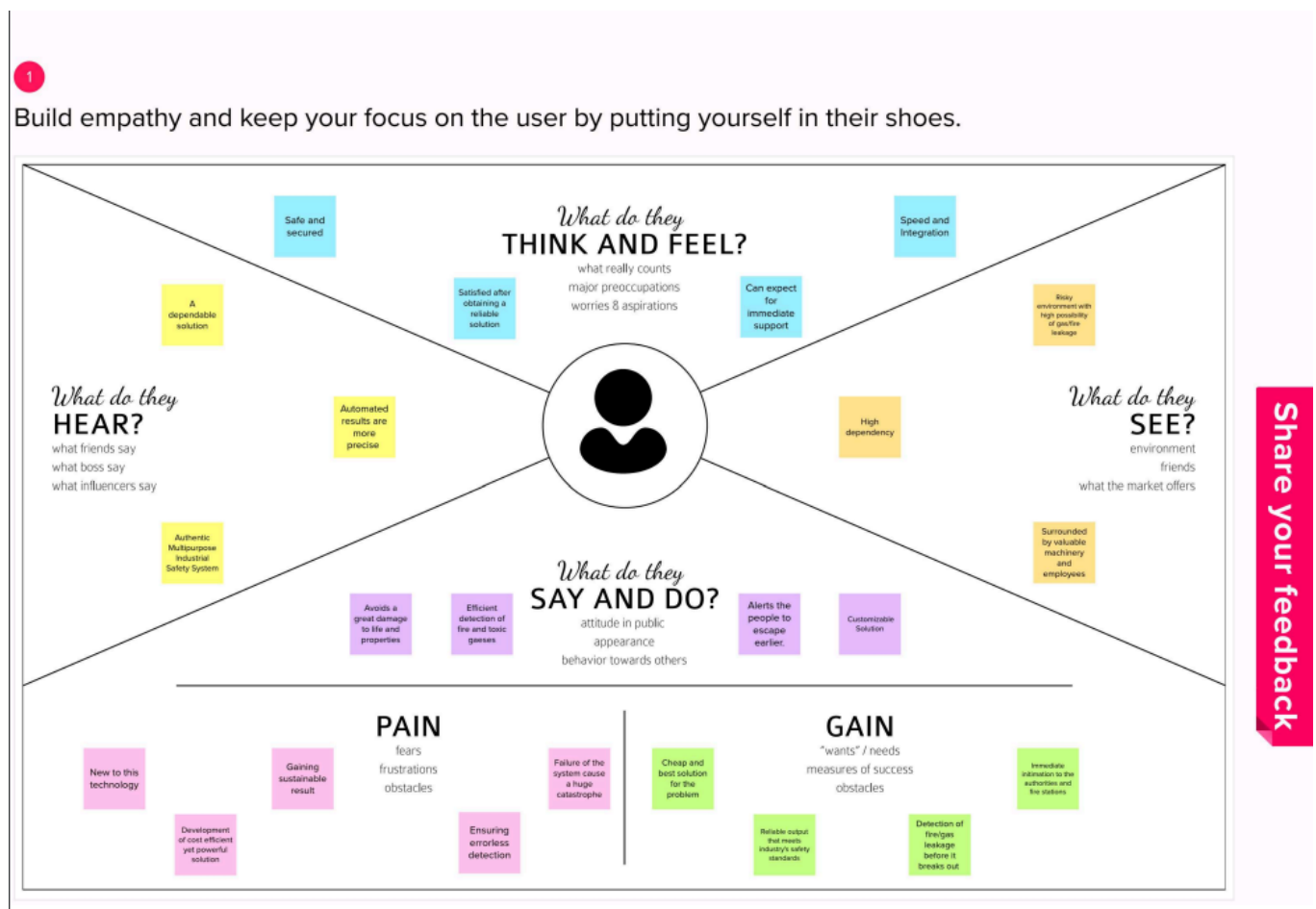
The fire management system can be used to assessing and controlling the fire risks.

- 1) If the temperature readings cross the threshold exhaust fans are powered ON.
- 2) If flame is detected sprinkles will be switched ON.
- 3) Alarms will be turned ON to alert the employees.
- 4) Message will be sent along with the location to fire station incase if the flame is uncontrollable.

PROBLEM STATEMENT DEFINITION:

An industry needs a way to avoid the possibilities of unexpected accidents due to fire/gss leakage in the working area by sensing, alerting and taking necessary measures, in order to avoid those accidents because it causes damage to valuable human lives, properties and machinery.

EMPATHY MAP CANVAS:



See photo



Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts, even if you're not sitting in the same room.

- 60 minutes to prepare
- 1 hour to collaborate
- 2.0 people recommended



22. [Share template feedback](#)

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

⌚ 60 minutes

45 minutes

- Team gathering**
Define who should participate in the session and send an invite. Share relevant information at pre-work ahead.
 - Set the goal**
Think about the problem you'll be focusing an solving in the brainstorming session.
 - Learn how to use the Facilitation Tools**
Use the Facilitation Superpowers to run a happy and productive session.
- [Open article](#)

[Open article](#)

1

What problem are you trying to solve? Frame your problem as a How Might It Be statement. This will be the focus of your investigation.

...of your



To set an smooth and productive work

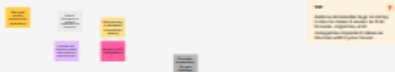
- | | |
|--|--|
|  Stay positive. |  Encourage with ideas. |
|  Reduce judgment. |  Listen to others. |
|  Use for volume. |  Hypothesis, be critical. |

Write down any ideas that come to mind that address your problem statement.



3

Take turns sharing your ideas while discussing similar or related notes as you go. In the last 10 minutes, give each cluster a sentence like below. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

 20 minutes

Your team should all be on the same page about what's important/moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

26 minutes



*You can request that we remove an image or poll to share with members of your company who might find it helpful.

- Share the mural**
Share a share link to the mural with stakeholders to keep them in the loop about the outcomes of the session.
- Export the mural**
Export a copy of the mural as a PDF or PNG to attach to emails, include in reports, or save in your drive.

 Energy Blueprint

- Define the components of a new idea or strategy**
[Open the template](#)
- Customer experience journey map**
Understand customer needs, motivations, and attitudes for an experience.
[Open the template](#)
- Strengths, weaknesses, opportunities & threats**
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
[Open the template](#)

[Share template as outline](#)

PROPOSED SOLUTION:

S.No	Parameter	Description
1	Problem statement	The Smart fire management system helps to detect the changes in the environment using gas sensor, flame sensor and temperature sensor and deals with the aftermath caused by any breakdown in the workspace.
2	Idea / Solution description	1) If the temperature readings cross the threshold exhaust fans are powered ON. 2) If flame is detected sprinkles will be switched ON. 3) Alarms will be turned ON to alert the employees. 4) Message will be sent along with the location to fire station in case if the flame is uncontrollable.
3	Novelty / Uniqueness	Usage of liquid Nitrogen. Liquid nitrogen will immediately vaporize causing a cooling effect and makes the site deprived of oxygen. Using water sprinklers after liquid nitrogen can put off even intense fire effectively.
4	Social impact / Customer satisfaction	1) Harmful gases can be purified and released into atmosphere 2) Cause of impact can be traced from the sensor data that can be analyzed to prevent future accidents.
5	Business model (Revenue model)	1) This is used to calculate the probability of ignition and spread of fires across a landscape. 2) This outcome allows for a better understanding of how changes in one aspects of management can affect other aspects of management.
6	Scalability of the solution	1) This can be implemented in all type of industries. 2) This design can be administered in restaurants which deals with high usage of fire and gas

PROBLEM SOLUTION FIT:

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) All sort of industrialist CS	6. CUSTOMER CONSTRAINTS CC The intense heat emitted from the fire breakout, toxic gas leakage, insecurities and fear caused from the incident prevent the customer from taking action.	5. AVAILABLE SOLUTIONS AS Availability of fire extinguishers, sprinklers and alerting system. Pros – Able to put out small fire breakout Cons – In case of large fire breakout the available system is ineffective. Automated but malfunctioning may occur sometimes.	Explore AS, differentiate
	2. JOBS-TO-BE-DONE / PROBLEMS J&P 1. Providing quick response (fire extinguish) for fire breakout. 2. An immediate alerting system.	9. PROBLEM ROOT CAUSE RC 1. Improper maintenance and handling of machineries. 2. Electrical short circuiting or overloading 3. Failure of machinery or other equipment. 4. Overheating of electrical or electronic appliances.	7. BEHAVIOUR BE Installation of proper fire and other safety management system to ensure the safety of their Industry.	
Focus on J&P, lap into BE, understand RC	3. TRIGGERS TR In need of mandatory safety system to be available in the Industry, hearing and reading about industrial fire breakout and gas leakage in the news.	10. YOUR SOLUTION SL Usage of automatic sprinklers along with liquid nitrogen is able to put off a large fire break out. Alerting the industrial workers and immediate intimation to the fire station. In case of toxic gas leakage, they are purified and released into the atmosphere.	8. CHANNELS of BEHAVIOUR CH 8.1 ONLINE Continuous monitoring of the safety management system. Signals are sent to sprinklers and alarm. 8.2 OFFLINE Immediate actions taken to the corresponding incident occurred. Switching ON of the respective actuators in need.	
4. EMOTIONS: BEFORE / AFTER EM Before – With availability of only extinguisher they feel panic, in secured and anxious while using it. After – With the help of our intelligent fire management system, since it is an automatic system they don't need to panic. They feel relaxed and able to handle the situation without anxiety.				

SOLUTION REQUIREMENTS:

Functional Requirements:

Following are the functional requirements of the proposed solution.

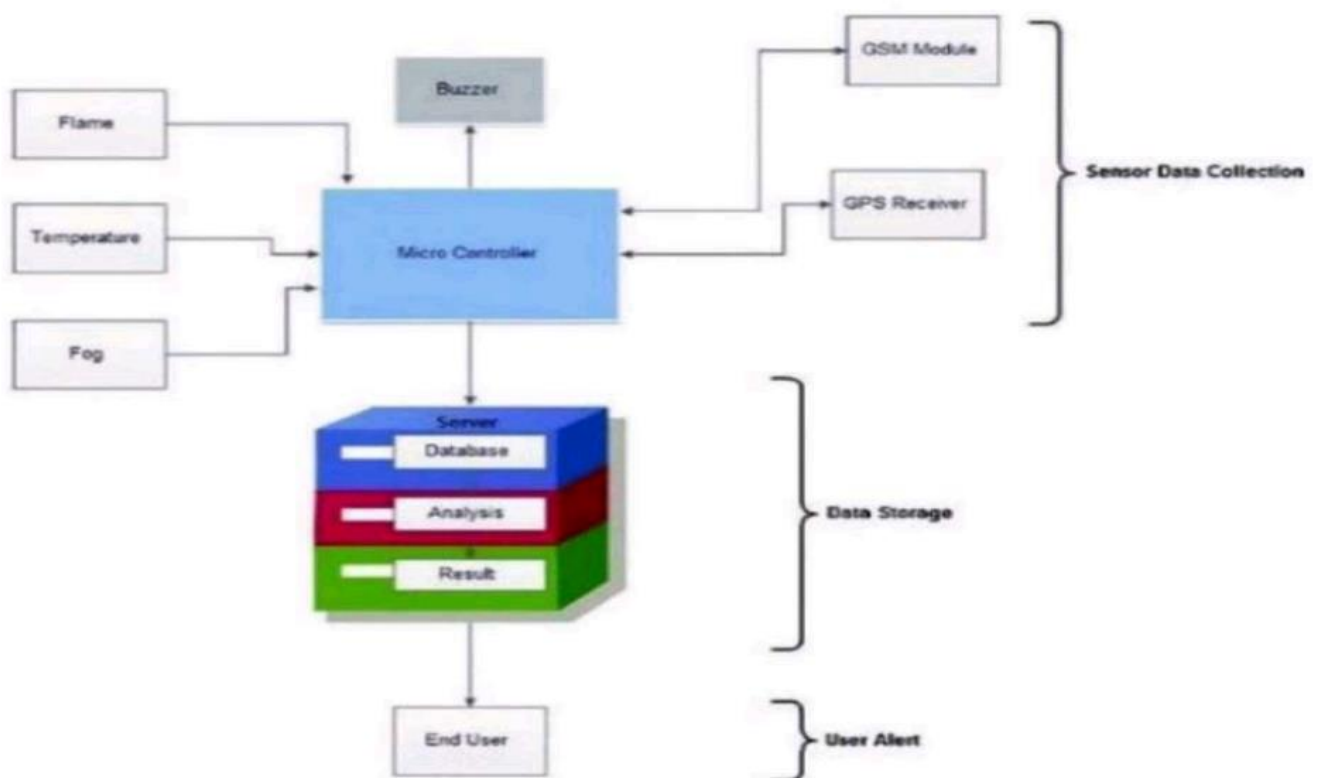
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Sensing function	Fire breakout has to be sensed by smoke detectors. Gas leakage has to be sensed by gas sensors.
FR-2	Alerting function	Blaring of alarms.
FR-3	Actuation function	Activation of sprinklers. Turning ON the exhaust Fan.
FR-4	Notification	Sending SMS with location to the fire station. Sending SMS to the authorities.

Non-functional Requirements:

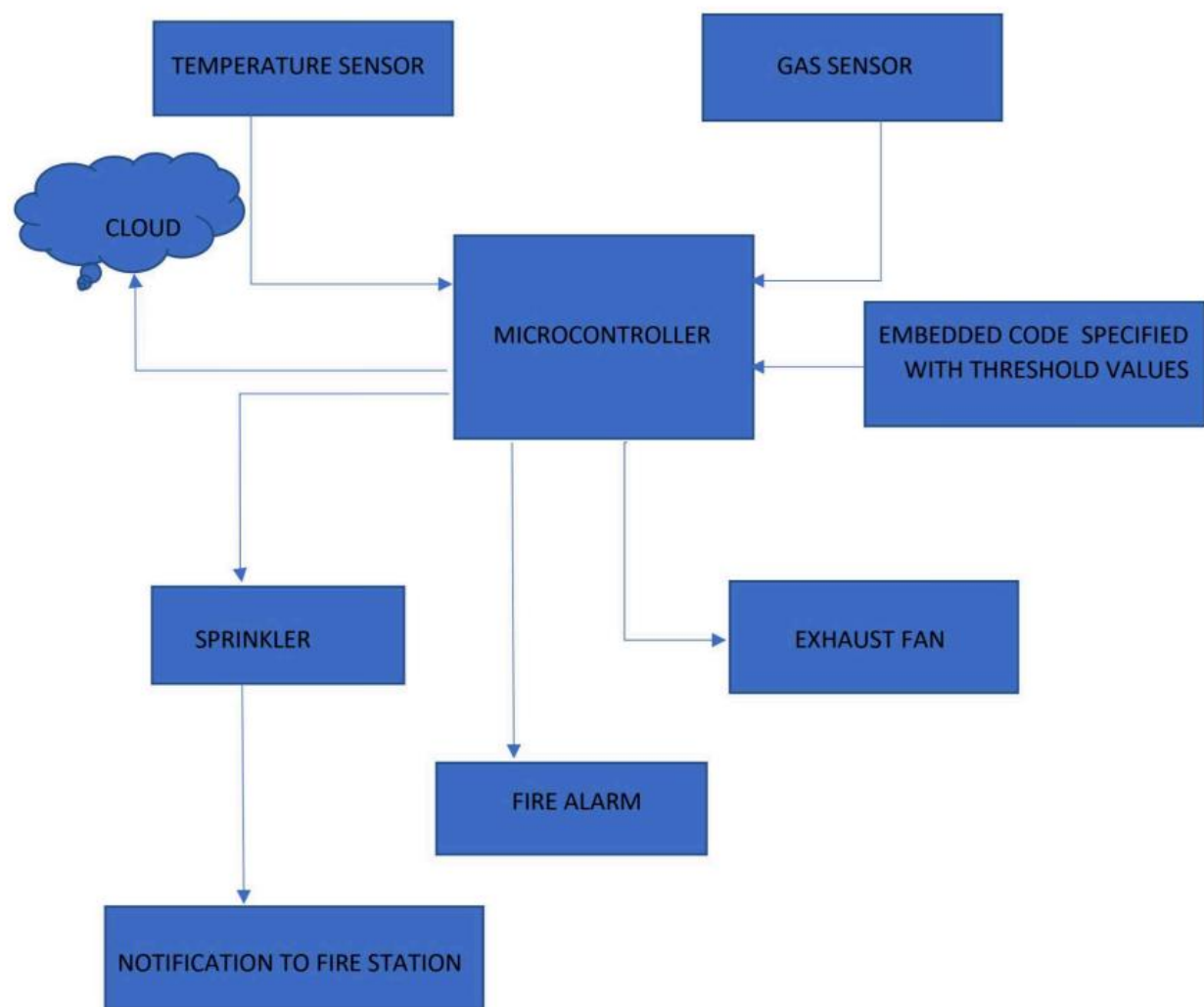
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Ease of use and longevity of the system.
NFR-2	Security	Software remains secured in the face of attacks.
NFR-3	Reliability	High accuracy.
NFR-4	Performance	Faster response.
NFR-5	Availability	Availability of the systems for institutions, restaurants and other public places
NFR-6	Scalability	It accommodates easy modification for various requirements

DATAFLOW DIAGRAMS:



SOLUTION & TECHNICAL ARCHITECTURE:



USER STORIES:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (industrialist)	Sensor system	USN-1	Smoke sensor detecting the fire breakdown.	I can get information about fire breakdown.	High	Sprint-1
		USN-2	Gas sensors detecting gas leakage.	I can receive information about hazardous gas leakage.	High	Sprint-1
	Alerting system	USN-3	Blaring of alarm.	Alert the employees to leave the building.	Medium	Sprint-1
	Actuation system	USN-4	Turning ON the sprinklers.	To put of fire at the initial stage.	Medium	Sprint-2
		USN-5	Switching ON exhaust fan	To filter the poisonous gas.	High	Sprint-2
Customer (company management)	Notification system	USN-6	Sending SMS to fire station along with current location.	To alert the fire station immediately after analysing its severity.	High	Sprint 3
		USN-7	Notlifying the management about the incident happened.	To be aware of breakdown.	Low	Sprint 4
	Cloud deployment	USN-8	Pushing data to the cloud.	To store and retrieve data for future requirements.	Low	Sprint 4

MILESTONES

1. Prerequisties

- » IBM Cloud Services
- » Software

2.Project Objectives

- » Abstract
- » Brainstorming

3.Create and Configure IBM Cloud Services

- » Create IBM Watson Iot Platform and Device
- » Create Node- Red Service
- » Create A Database in Cloudant DB

4.Develop the Python Script

- » Develop A Python Script

5. Develop A Web Application Using Node-RED Service.

- » Develop The Web Application Using Node-RED

6.Ideation Phase

- » Literature Survey on The Selected Project & Information Gathering
- » Prepare Empathy Map
- » Ideation

7.Project Design Phase -1

- » Proposed Solution
- » Prepare Solution Fit
- » Solution Architecture

8.Project Design Phase -2

- » Customer journey
- » Functional Requirement
- » Data Flow Diagram
- » Technology Architecture

9.Project planning Phase

- » Prepare Milestones & Activity List
- » Sprint Delivery Plan

10.Project Development Phase

- » Project Development-Delivery Of Sprint-1

- » Project Development-Delivery Of Sprint-2
- » Project Development-Delivery Of Sprint-3
- » Project Development-Delivery Of Sprint-4

TECHNOLOGY STACK:

COMPONENTS AND TECHNOLOGIES:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson S I I service
4.	Application Logic-3	Logic for a process in the application	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration:	Local, Cloud Foundry, Kubernetes, etc.

APPLICATION CHARACTERISTICS:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source Frameworks used	Technology of Opensource framework
2.	Security Implementations	List all the security / access controls implemented, use of firewalls etc.	e.g., SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Justify the scalability of architecture (3 — tier, Micro-services)	Technology used
4.	Availability	Justify the availability of application (e.g., use of load balancers, distributed servers etc.)	Technology used
5.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Technology used

SPRINT DELIVERY PLAN:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Sensing	USN-1	Sensing the environment using the sensors.	3	High	Kaviya
	Operating	USN-2	Turning on the exhaust fan as well as the fire sprinkler system in cause of fire and gas leakage.	3	Medium	Jigendhra Kaviya
Sprint-2	Sending collected data to the IBM Watson platform	USN-3	Sending the data of the Sensors to the IBM Watson.	3	High	Preethi Karishma
	Node red	USN-4	Sending the data from the IBM Watson to the Node red.	3	High	Karishma Jigendhra

Sprint-3	Storing of sensor data	USN-5	Storing in Cloudant database.	2	Medium	Kaviya Preethi
	Registration	USN-6	Entering my email and password to verify authentication process.	1	Medium	Jigendhra Karishma
	Web UI	USN-7	Monitors the situation of the environment which displays sensor information.	3	High	Kaviya Preethi
Sprint-4	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	Kaviya Preethi Jigendhra Karishma
	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	Kaviya Preethi Jigendhra Karishma
	Testing	USN-10	Testing of project and Final Deliverables.	1	Low	Kaviya Preethi Jigendhra Karishma

SPRINT 1:

```
#include <WiFi.h>//library for wifi
#include <PubSubClient.h>//library for MQTT
#include "DHT.h"// Library for dht11
#define DHTPIN 15 // what pin we're connected to
#define DHTTYPE DHT22 // define type of sensor DHT 11
#define LED 2
```

```

DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin and typr of dht connected

void callback(char* subscribetopic, byte* payload, unsigned int payloadLength);

//-----credentials of IBM Accounts-----

#define ORG "gh6uoi" //IBM ORGANITION ID

#define DEVICE_TYPE "trial1" //Device type mentioned in ibm watson IOT Platform

#define DEVICE_ID "1234abcd" //Device ID mentioned in ibm watson IOT Platform

#define TOKEN "12345678" //Token

String data3;

float h, t;

//----- Customise the above values -----

char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name

char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform and format
in which data to be send

char subscribetopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT command type AND
COMMAND IS TEST OF FORMAT STRING

char authMethod[] = "use-token-auth"; // authentication method

char token[] = TOKEN;

char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID; //client id

//-----

WiFiClient wifiClient; // creating the instance for wificlient

PubSubClient client(server, 1883, callback ,wifiClient); //calling the predefined client id by passing
parameter like server id,portand wificredential

void setup() // configureing the ESP32

{
  Serial.begin(115200);

  dht.begin();

  pinMode(LED,OUTPUT);

  delay(10);

  Serial.println();

  pinMode(mq2, INPUT);

```

```

pinMode (flame_sensor_pin , INPUT ); // declaring sensor pin as input pin for Arduino
pinMode(BUZZER_PIN, OUTPUT);
wificonnect();
mqttconnect();
}
void loop()// Recursive Function
{
t = dht.readTemperature();
Serial.print("temp:");
Serial.println(t);
if(t > 60)
{
Serial.println("Alert");
digitalWrite(BUZZER_PIN, HIGH); // turn on
}
else
{
digitalWrite(BUZZER_PIN, LOW); // turn on
}
int gassensorAnalogmq2 = analogRead(mq2);
Serial.print("mq2 Gas Sensor: ");
Serial.print(gassensorAnalogmq2);
if (gassensorAnalogmq2 > 1500)
{
Serial.println("mq2Gas");
Serial.println("Alert");
}
else
{
Serial.println("No mq2Gas");
}
}

```

```

}

flame_pin = digitalRead ( flame_sensor_pin ) ; // reading from the
sensor if (flame_pin == LOW ) // applying condition
{
Serial.println ( " ALERT: FLAME DETECTED" ) ;
digitalWrite ( BUZZER_PIN , HIGH ) ;// if state is high, then turn high the
BUZZER }
else
{
Serial.println ( " NO FLAME DETECTED " ) ;
digitalWrite ( BUZZER_PIN , LOW ) ; // otherwise turn it low
}

PublishData(t, gassensorAnalogmq2, flame_pin);
delay(1000);
if (!client.loop()) {
  mqttconnect();
}
}

/*.....retrieving to Cloud.....*/

void PublishData(float t, float gassensorAnalogmq2 , int flame_pin ) {
  mqttconnect();//function call for connecting to ibm

  /*
  creating the String in in form JSon to update the data to ibm cloud
  */
  String payload = "{\"temp\":";
  payload += t;
  payload += "," "\"gasvalue\":";
  payload += gassensorAnalogmq2;
  payload += "," "\"flame\":";
  payload += flame_pin;

```

```

payload += "}";

Serial.print("Sending payload: ");
Serial.println(payload);
if (client.publish(publishTopic, (char*) payload.c_str())) {
    Serial.println("Publish ok");// if it sucessfully upload data on the cloud then it will print publish ok
    in Serial monitor or else it will print publish failed
} else {
    Serial.println("Publish failed");
}
}

void mqttconnect() {
    if (!client.connected()) {
        Serial.print("Reconnecting client to ");
        Serial.println(server);
        while (!client.connect(clientId, authMethod, token)) {
            Serial.print(".");
            delay(500);
        }
        initManagedDevice();
        Serial.println();
    }
}

void wificonnect() //function defination for wificonnect
{
    Serial.println();
    Serial.print("Connecting to ");
    WiFi.begin("Wokwi-GUEST", "", 6);//passing the wifi credentials to establish the connection
    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++) {
    //Serial.print((char)payload[i]);
    data3 += (char)payload[i];
  }

  Serial.println("data: "+ data3);

  if(data3=="lighton")
  {
    Serial.println(data3);
    digitalWrite(LED,HIGH);
  }
}

```

```

else
{
Serial.println(data3);
digitalWrite(LED,LOW);
}

data3="";
}

```

SPRINT 2:

INTERFACING CLOUD WITH INTERNET OF THINGS PLATFORM:

1. CREATING SIMULATION

IBM Watson IoT Platform

?

kari.1914119@gct.ac.in

ID: gh6ud

Browse

Action

Device Types

Interfaces

Add Device

<input type="checkbox"/>	Device ID	Status	Device Type	Class ID	Date Added	Descriptive Location
>	1234abcd	Disconnected	trial1	Device	7 Nov 2022 21:44	
>	abcd_1	Disconnected	abcd	Device	15 Nov 2022 17:52	
▼	fire123	Disconnected	fire	Device	17 Nov 2022 20:02	→ ...

Identity

Device Information

Recent Events

State

Logs

×

Device ID

fire123

Device Type

fire

Date Added

17 Nov 2022 20:02

Added By

kari.1914119@gct.ac.in

Connection Status

Disconnected

◀

Device Type: fire

▼

Events 1

New event type

Event type name

event_1

Send

Schedule

1

Every Minute

Payload

Specify the event payload in the editor window or by uploading a CSV file.

0

{

1

"randomNumber": random(0, 100),

2

"temperature": random(30, 150),

3

"Gas": random(60, 200),

4

"Flame": random(60, 200)

5

}

6

Cancel

Save

IBM Watson IoT Platform

?

karl.1914119@gct.ac.in

ID: gh6-ujl

Browse

Action

Device Types

Interfaces

Add Device

Identity

Device Information

Recent Events

State

Logs

The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
IoTSensor	{"temp":18,"Gas":178,"Flame":165}	json	a few seconds ago
IoTSensor	{"temp":40,"Gas":96,"Flame":192}	json	a few seconds ago
IoTSensor	{"temp":20,"Gas":139,"Flame":66}	json	a few seconds ago
IoTSensor	{"temp":86,"Gas":146,"Flame":67}	json	a few seconds ago
IoTSensor	{"temp":34,"Gas":174,"Flame":187}	json	a few seconds ago

2. ADDING BOARDS:

Card source data

fire123

Card preview

Card information

Create Gauge Card

Specify the data source for the card

Devices

Search for card data sources using the filter:

	Device ID	Device Type
<input type="radio"/>	1234abcd	trial1
<input type="radio"/>	abcd_1	abcd
<input checked="" type="radio"/>	fire123	fire
<input type="radio"/>	trial1_1	trial1
<input type="radio"/>	trial1_2	trial1

Next

Card source data

fire123

Card preview

Card information

Create Gauge Card

Connect data set

temperature

Event

event_1

Property

temperature

Name

temperature

Type

Number

Unit

°C

Min

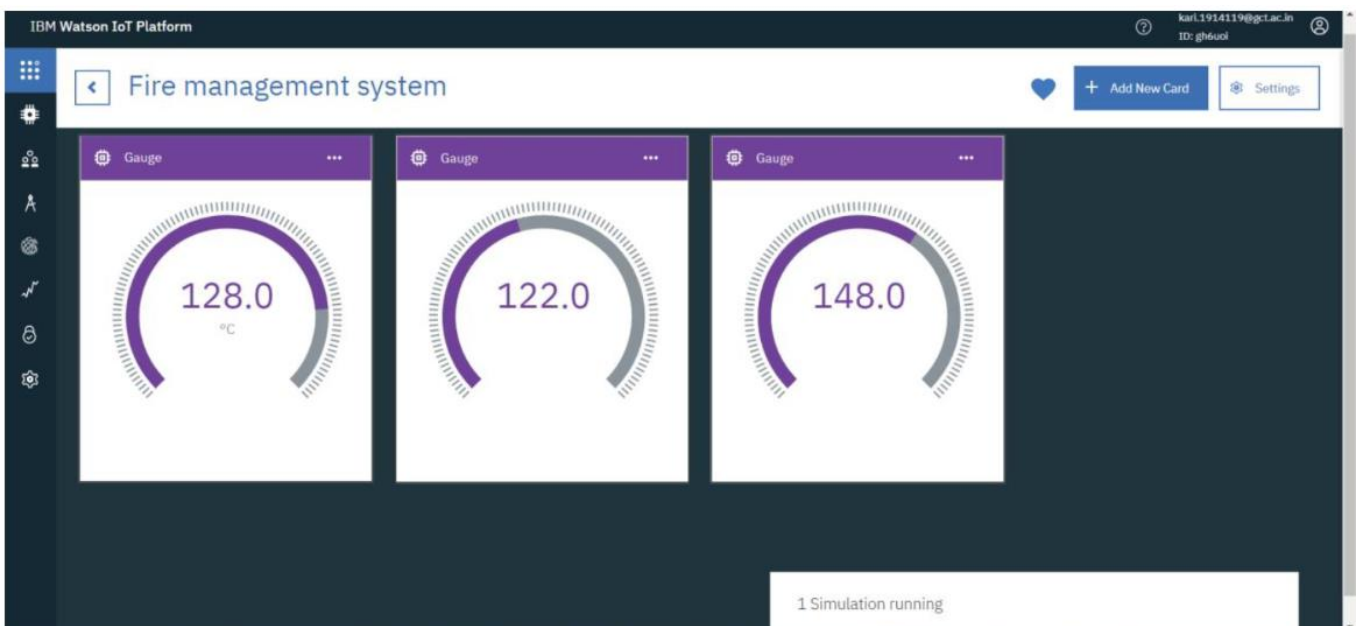
30

Max

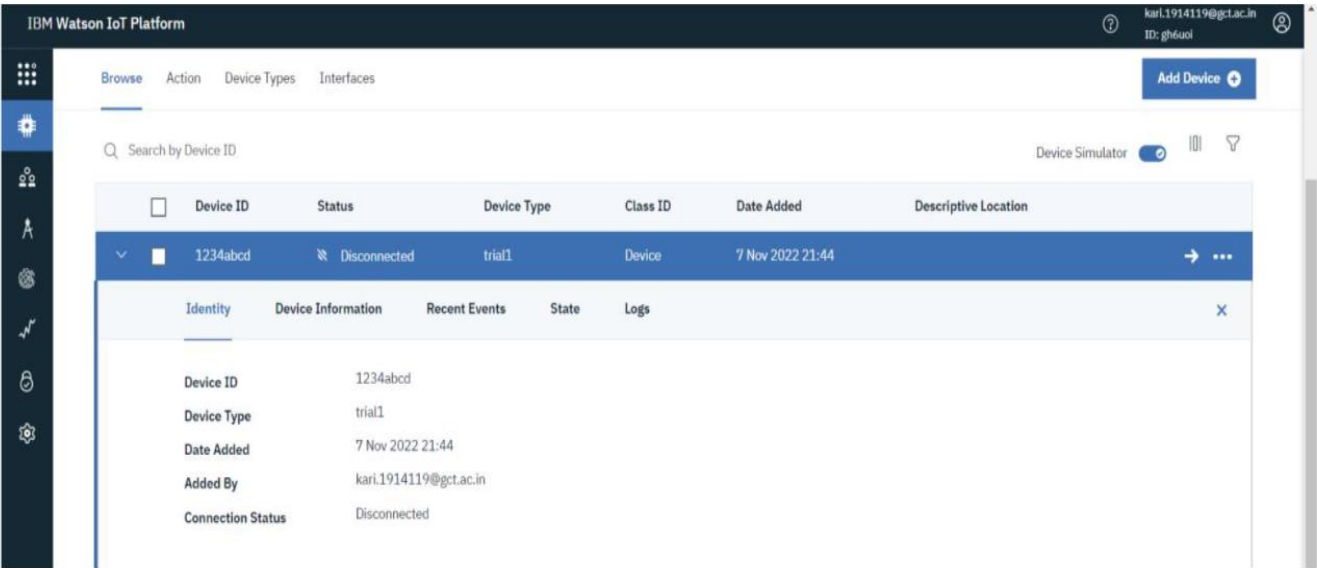
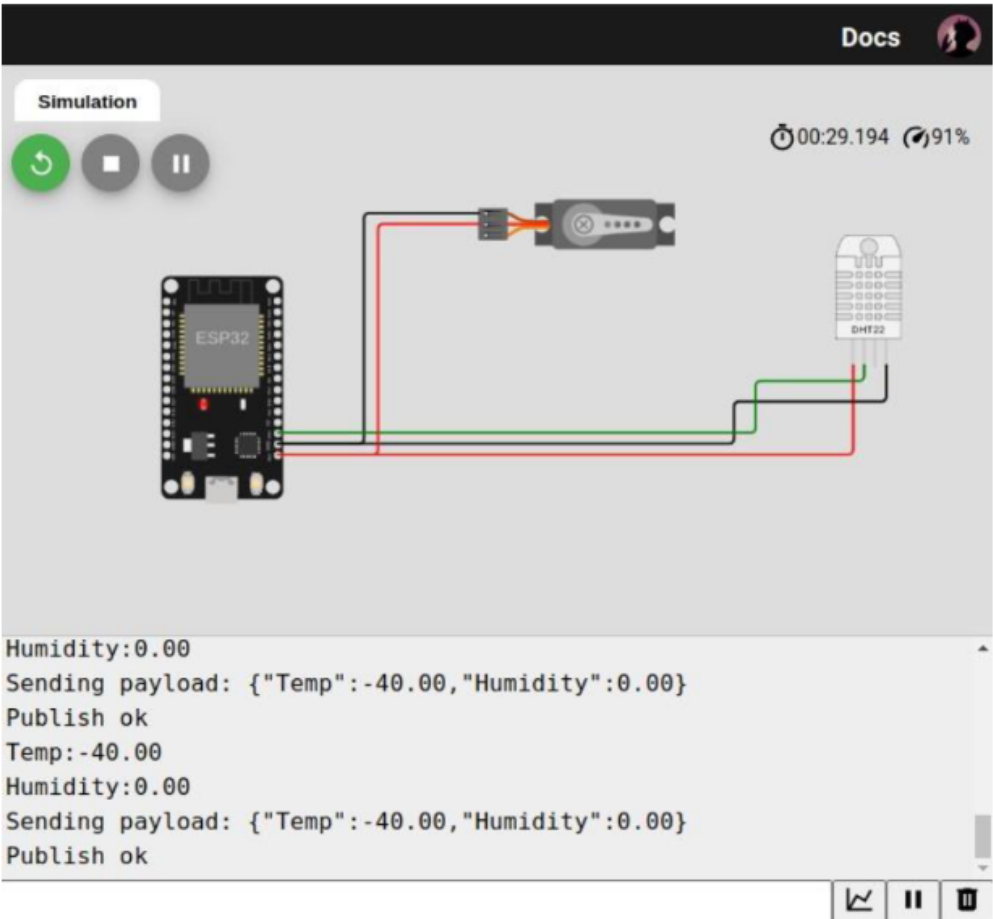
150

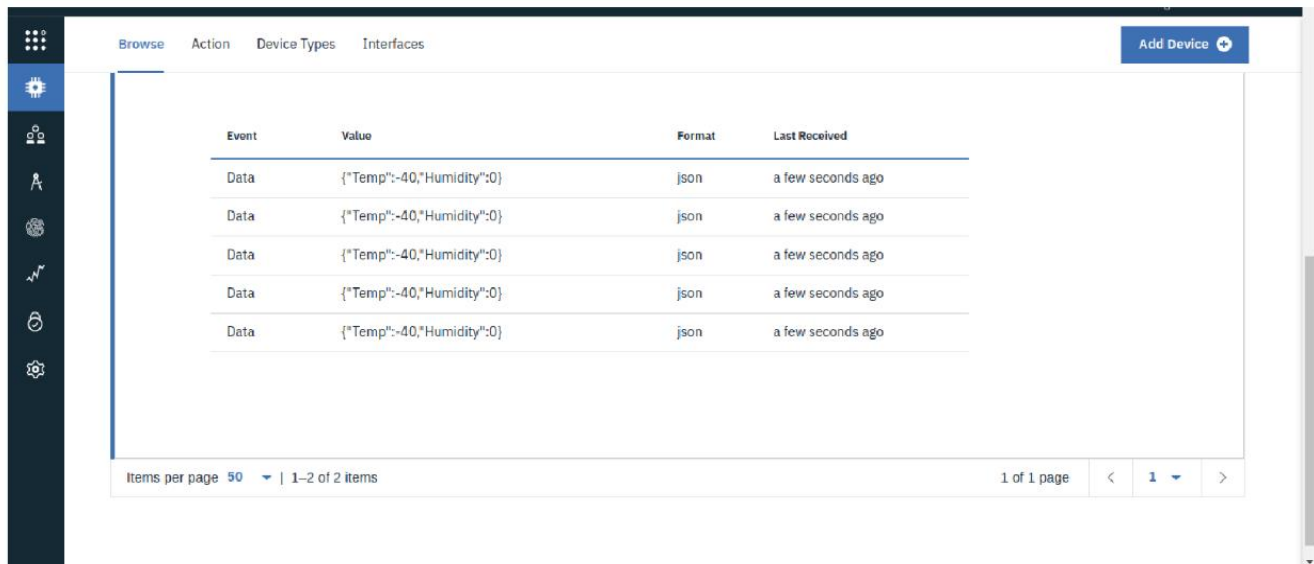
Back

Next



WOKWI SIMULATION:





The screenshot shows the IBM Watson IoT Platform interface. On the left is a dark sidebar with various icons. The main area has a top navigation bar with 'Browse', 'Action', 'Device Types', and 'Interfaces'. A blue 'Add Device' button is in the top right. Below the navigation bar is a table with the following data:

Event	Value	Format	Last Received
Data	{"Temp":-40,"Humidity":0}	json	a few seconds ago
Data	{"Temp":-40,"Humidity":0}	json	a few seconds ago
Data	{"Temp":-40,"Humidity":0}	json	a few seconds ago
Data	{"Temp":-40,"Humidity":0}	json	a few seconds ago
Data	{"Temp":-40,"Humidity":0}	json	a few seconds ago

At the bottom of the table, there is a pagination bar showing 'Items per page 50', '1-2 of 2 items', and '1 of 1 page'.

SPRINT- 3:

PYTHON SCRIPT:

```
import time

import sys

import ibmiotf.application

import ibmiotf.device

import random

# Provide your IBM Watson Device Credentials

organization = "gh6uoi"

deviceType = "fire"

deviceId = "fire123"

authMethod = "token"

authToken = "0123456789"

# 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 ()
```

```
def myCommandCallback(cmd) :
```

```
    print ( "Command received: %s" % cmd.data['command'] )
```

```
    status = cmd.data['command']
```

```
def myCommandCallback(cmd) :
```

```
    print ( "Command received: %s" % cmd.data['command'] )
```

```
    status = cmd.data['command']
```

```
def myCommandCallback(cmd) :
```

```
    print ( "Command received: %s" % cmd.data['command'] )
```

```
    status = cmd.data['command']
```

```
while True :
```

```
    # Get Sensor Data from DHT11
```

```
    temp = random.randint ( 0, 100 )
```

```
    gas = random.randint ( 60, 200 )
```

```
    flame = random.randint ( 60, 200 )
```

```
    data = { 'temp' : temp, 'Gas' : gas, 'Flame': flame }
```

```
    # print data
```

```
    def myOnPublishCallback() :
```

```
        print ( "Published Temperature = %s C" % temp, "Gas = %s %" % gas, "Flame = %s %" %  
flame, "to IBM Watson")
```

```
        success = deviceCli.publishEvent ( "IoTSensor", "json", data, qos = 0, on_publish =  
myOnPublishCallback )
```

```
        if not success :
```

```
    print ( "Not connected to IoT" )

time.sleep ( 1 )

deviceCli.commandCallback = myCommandCallback

# Initialize GPIO

if temp > 50 :

    print ( "buzzer is on" )

else :

    print ( "buzzer is off" )

if flame > 100 :

    print ( "sprinklers are on" )

else :

    print ( "sprinklers are off" )

if gas>100:

    print ( "exhaust fan is on" )

else :

    print ( "exhaust fan is off" )

# Disconnect the device and application from the cloud

deviceCli.disconnect ()
```

OUTPUT OF PYTHON CODE:

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Published Temperature = 11 C Gas = 100 % Flame = 70 % to IBM Watson
buzzer is off
sprinklers are off
exhaust fan is on
Published Temperature = 93 C Gas = 161 % Flame = 154 % to IBM Watson
buzzer is on
sprinklers are on
exhaust fan is on
Published Temperature = 7 C Gas = 187 % Flame = 170 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is on
Published Temperature = 97 C Gas = 104 % Flame = 176 % to IBM Watson
buzzer is on
sprinklers are on
exhaust fan is on
Published Temperature = 74 C Gas = 133 % Flame = 91 % to IBM Watson
buzzer is on
sprinklers are off
exhaust fan is on
Published Temperature = 25 C Gas = 191 % Flame = 107 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is on
Published Temperature = 73 C Gas = 174 % Flame = 112 % to IBM Watson
buzzer is on
sprinklers are on
exhaust fan is on
Published Temperature = 10 C Gas = 91 % Flame = 142 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is off
Published Temperature = 56 C Gas = 69 % Flame = 128 % to IBM Watson
buzzer is on
sprinklers are on
exhaust fan is off
Published Temperature = 30 C Gas = 91 % Flame = 167 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is off
Published Temperature = 42 C Gas = 140 % Flame = 172 % to IBM Watson
Ln: 27 Col: 0
```

```
newfire.py - C:\Users\DELL\AppData\Local\Programs\Python\Python37\newfire.py (3.7.4)
File Edit Format Run Options Window Help
import time
import sys
import ibmiotf.application
import ibmiotf.device
import random

# Provide your IBM Watson Device Credentials
organization = "gh6uoi"
deviceType = "fire"
deviceId = "fire123"
authMethod = "token"
authToken = "0123456789"

# print(cmd)

try :
    deviceOptions = {"org" : organization, "type" : deviceType, "id" : deviceId,
                    "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
deviceCli.connect ()

def myCommandCallback(cmd) :
    print ( "Command received: %s" % cmd.data['command'] )
    status = cmd.data['command']

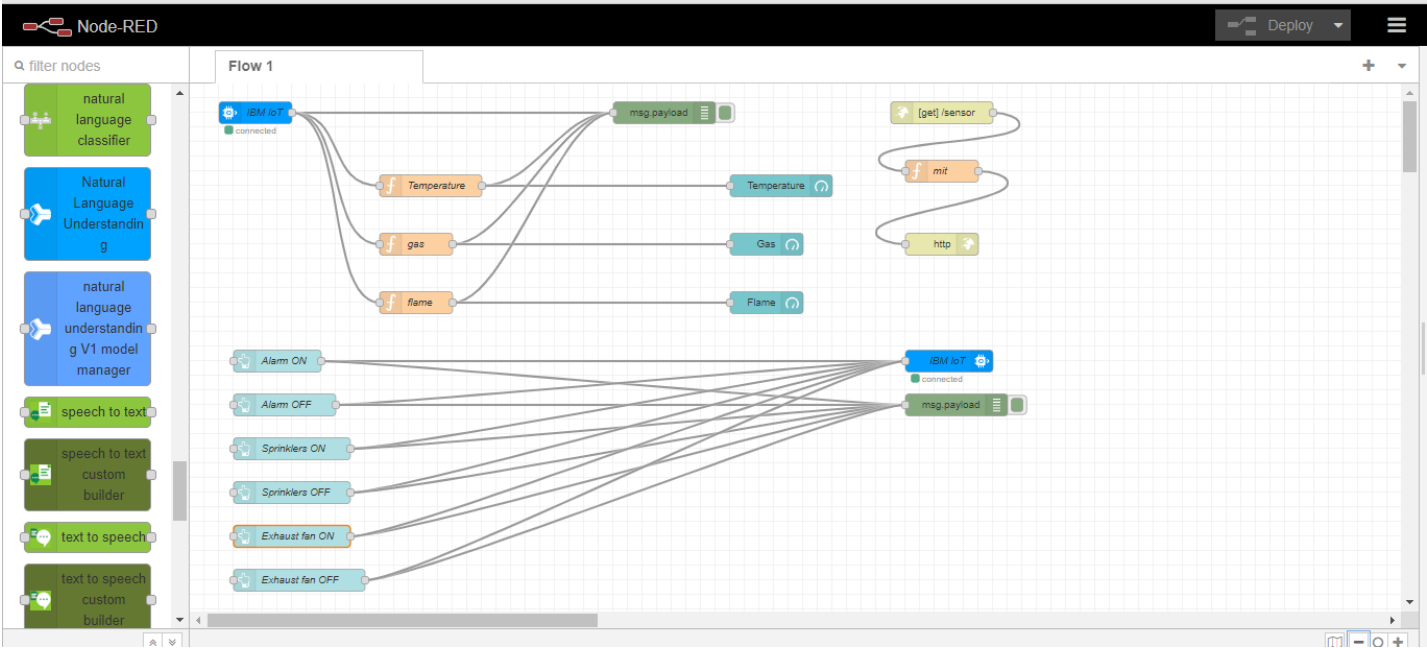
def myCommandCallback(cmd) :
    print ( "Command received: %s" % cmd.data['command'] )
    status = cmd.data['command']

def myCommandCallback(cmd) :
    print ( "Command received: %s" % cmd.data['command'] )
    status = cmd.data['command']

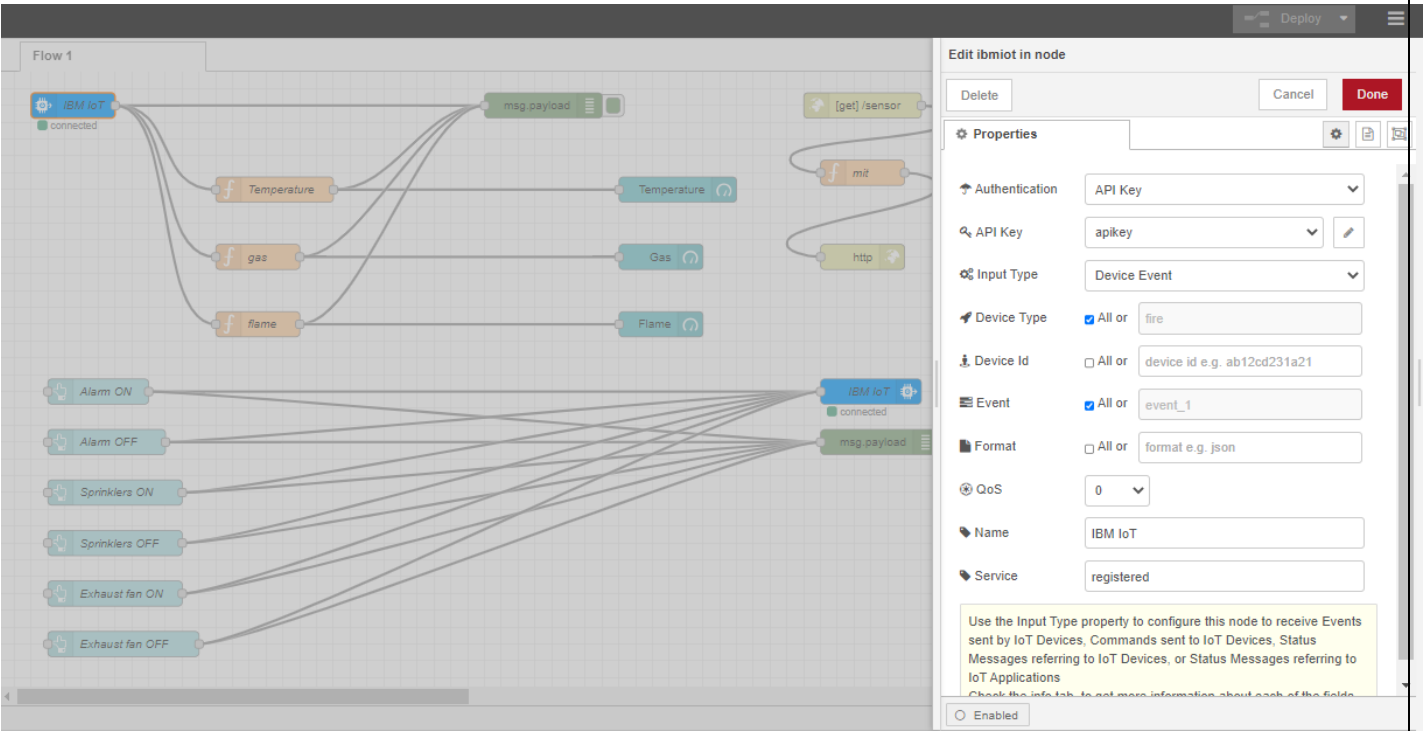
while True :
    # Get Command Data from IBM Watson

Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:\Users\DELL\AppData\Local\Programs\Python\Python37\newfire.py ==
2022-11-19 20:06:16,476 ibmiotf.device.Client INFO Connected successfully: digh6uoi:fire:fire123
Published Temperature = 30 C Gas = 118 % Flame = 102 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is on
Published Temperature = 97 C Gas = 180 % Flame = 95 % to IBM Watson
buzzer is on
sprinklers are off
exhaust fan is on
Published Temperature = 94 C Gas = 165 % Flame = 128 % to IBM Watson
buzzer is on
sprinklers are on
exhaust fan is on
Published Temperature = 42 C Gas = 69 % Flame = 181 % to IBM Watson
buzzer is off
sprinklers are on
exhaust fan is off
Published Temperature = 70 C Gas = 107 % Flame = 91 % to IBM Watson
Ln: 1 Col: 1
```

NODE RED:



IBMIOT IN NODE:



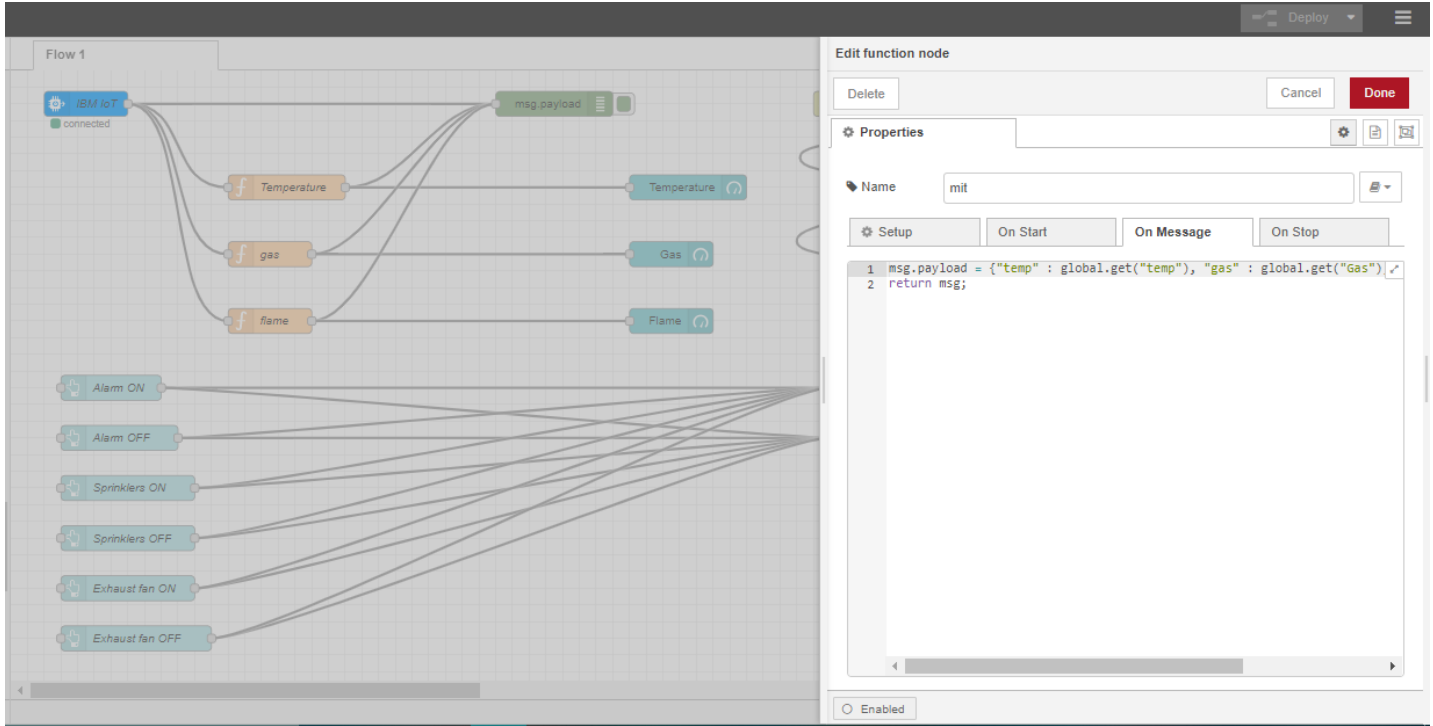
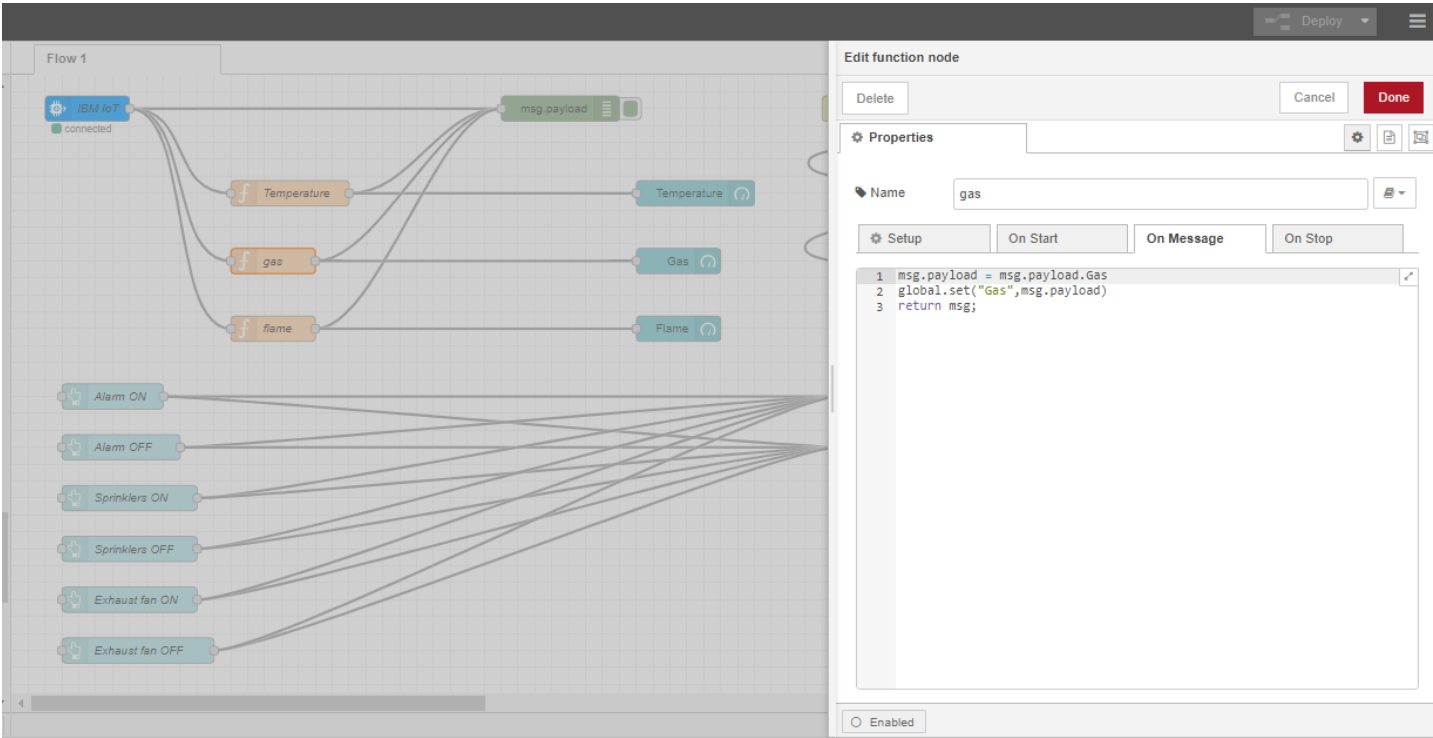
IBMIOT OUT NODE:

The screenshot shows the Node-RED web interface. On the left, a flow named 'Flow 1' is visible, containing an 'IBM IoT' node (connected), a 'msg.payload' node, and several function nodes labeled 'Temperature', 'gas', and 'flame'. These function nodes are connected to corresponding output nodes: 'Temperature', 'Gas', and 'Flame'. Below these, there are six button nodes: 'Alarm ON', 'Alarm OFF', 'Sprinklers ON', 'Sprinklers OFF', 'Exhaust fan ON', and 'Exhaust fan OFF'. All these button nodes are connected to the 'IBM IoT' node. On the right, the 'Edit ibmiot out node' configuration panel is open. It includes a 'Delete' button, 'Cancel', and 'Done' buttons. The 'Properties' section contains the following fields: Authentication (API Key), API Key (apikey), Output Type (Device Command), Device Type (fire), Device Id (fire123), Command Type (cmd), Format (json), Data (data), QoS (0), Name (IBM IoT), and Service (registered). A note at the bottom states: 'Note: If there is a property in the message that corresponds to any of the values entered above, then the property in the message takes'. At the very bottom, there is an 'Enabled' checkbox.

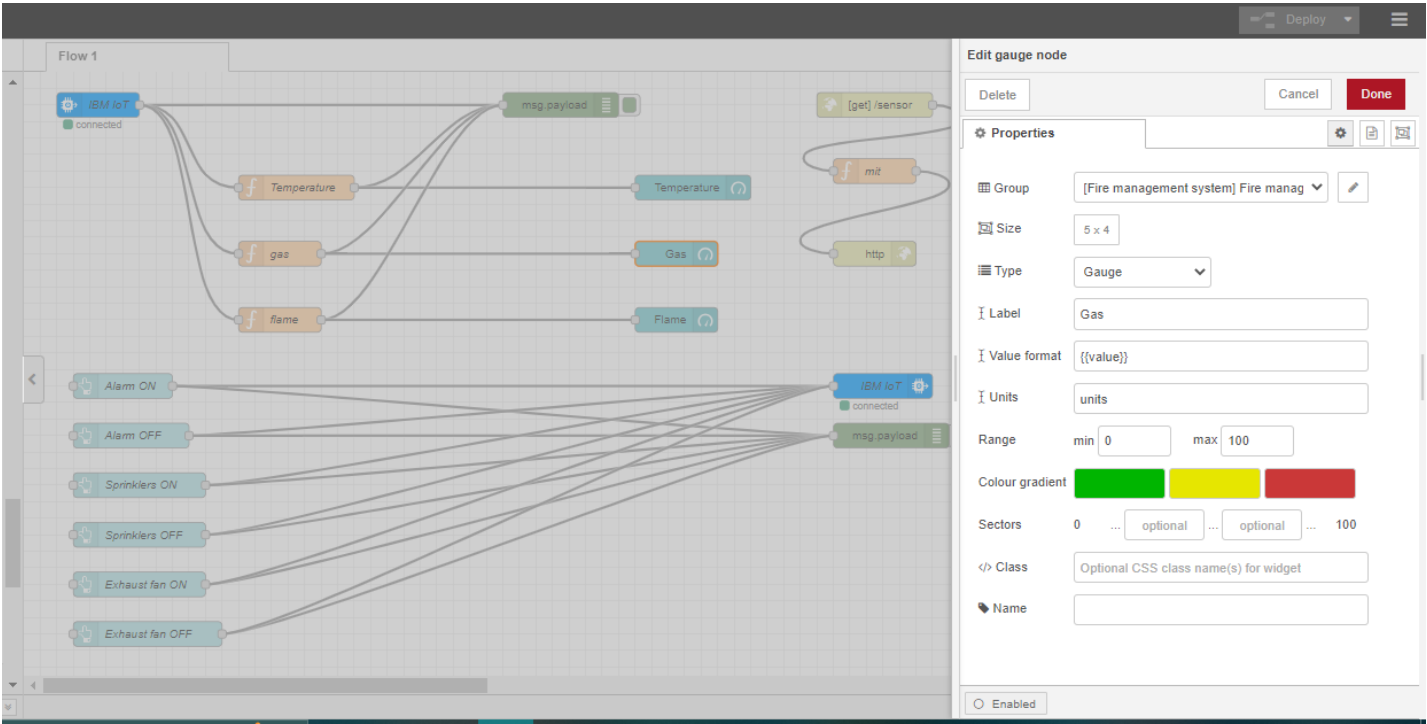
BUTTON NODE:

The screenshot shows the Node-RED web interface, similar to the one above. The flow 'Flow 1' is the same. The 'Edit button node' configuration panel is open on the right. It includes 'Delete', 'Cancel', and 'Done' buttons. The 'Properties' section contains: Group ([Fire management system] Fire man), Size (4 x 1), Icon (optional icon), Label (Alarm ON), Tooltip (optional tooltip), Color (optional text/icon color), and Background (optional background color). The 'When clicked, send:' section includes a Payload field with a dropdown menu showing '["command": "buzzer is on"]' and a Topic field with a dropdown menu showing 'msg. topic'. There is a checkbox for 'If msg arrives on input, emulate a button click:' and a 'Class' field with the value 'Optional CSS class name(s) for widget'. At the bottom, there is an 'Enabled' checkbox.

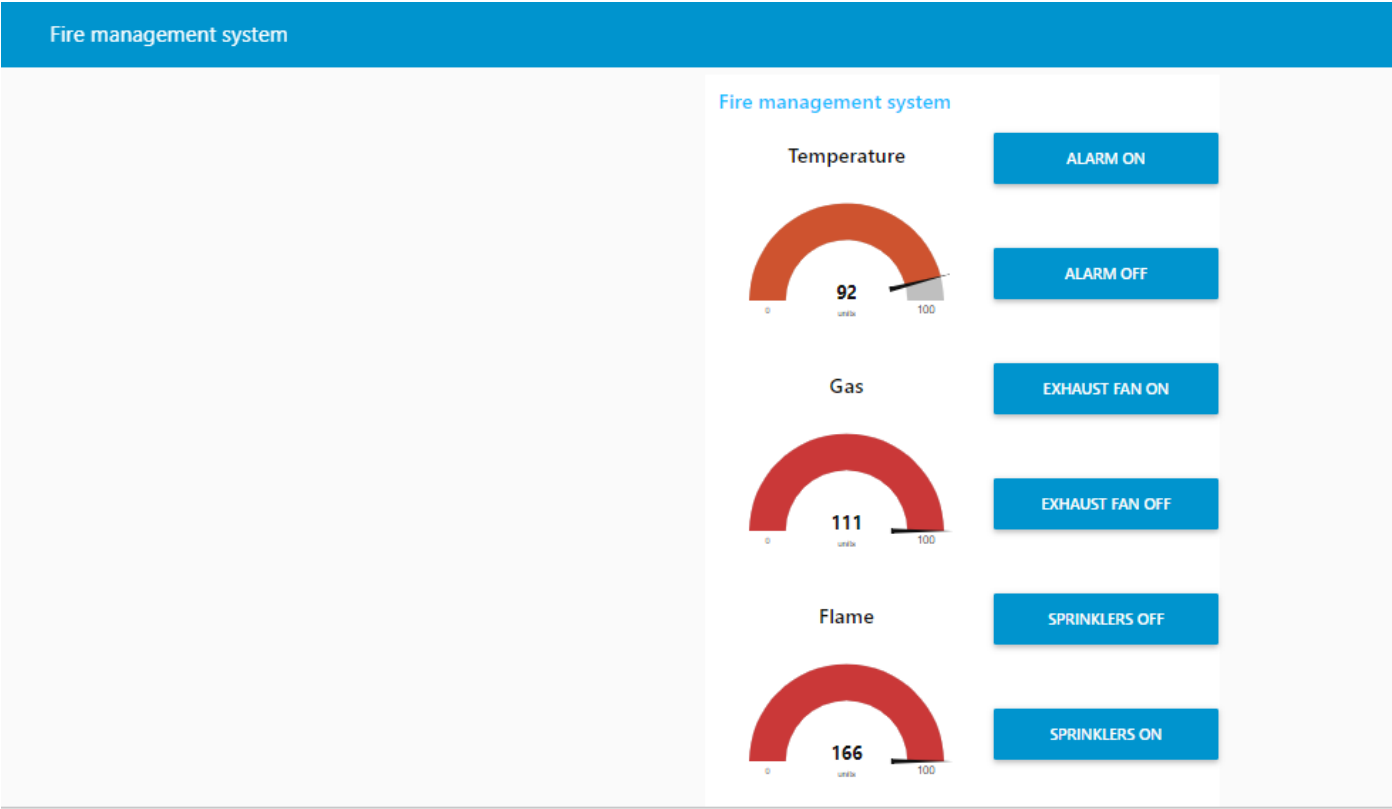
FUNCTION NODE:



GAUGE NODE:



NODE RED WEB UI:



CLOUD INTERFACING WITH NODE RED:

IBM Watson IoT Platform

kari.1914119@gct.ac.in
ID: gh6uoi

Browse Action Device Types Interfaces Add Device +

Identity Device Information **Recent Events** State Logs

The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
event_1	{"randomNumber":60,"temp":63,"gas":155,"fla...	json	a few seconds ago
event_1	{"randomNumber":85,"temp":64,"gas":90,"flame...	json	a few seconds ago
event_1	{"randomNumber":25,"temp":82,"gas":104,"fla...	json	a few seconds ago
event_1	{"randomNumber":53,"temp":1,"gas":126,"flame...	json	a few seconds ago
event_1	{"randomNumber":74,"temp":54,"gas":143,"fla...	json	a few seconds ago

1 Simulation running

IBM Watson IoT Platform

kari.1914119@gct.ac.in
ID: gh6uoi

Browse Action Device Types Interfaces Add Device +

Device ID	Status	Device Type	Class ID	Date Added
fire123	Disconnected	fire	Device	Nov 17, 2022 8:02 PM

Identity Device Information **Recent Events** State Logs

The recent events listed show the live stream of data that is coming and going from this device.

Event	Value	Format	Last Received
event_1	{"randomNumber":74,"temp":75,"gas":109,"fla...	json	a few seconds ago
event_1	{"randomNumber":30,"temp":30,"gas":125,"fla...	json	a few seconds ago
event_1	{"randomNumber":30,"temp":45,"gas":115,"fla...	json	a few seconds ago
event_1	{"randomNumber":19,"temp":24,"gas":116,"fla...		
event_1	{"randomNumber":32,"temp":26,"gas":145,"fla...		

1 Simulation running

CLOUDANT:

< db
Document ID ▾
 Options
{ } JSON

- All Documents +
- Query
- Permissions
- Changes
- Design Documents +

Table
Metadata
 JSON

Create Document

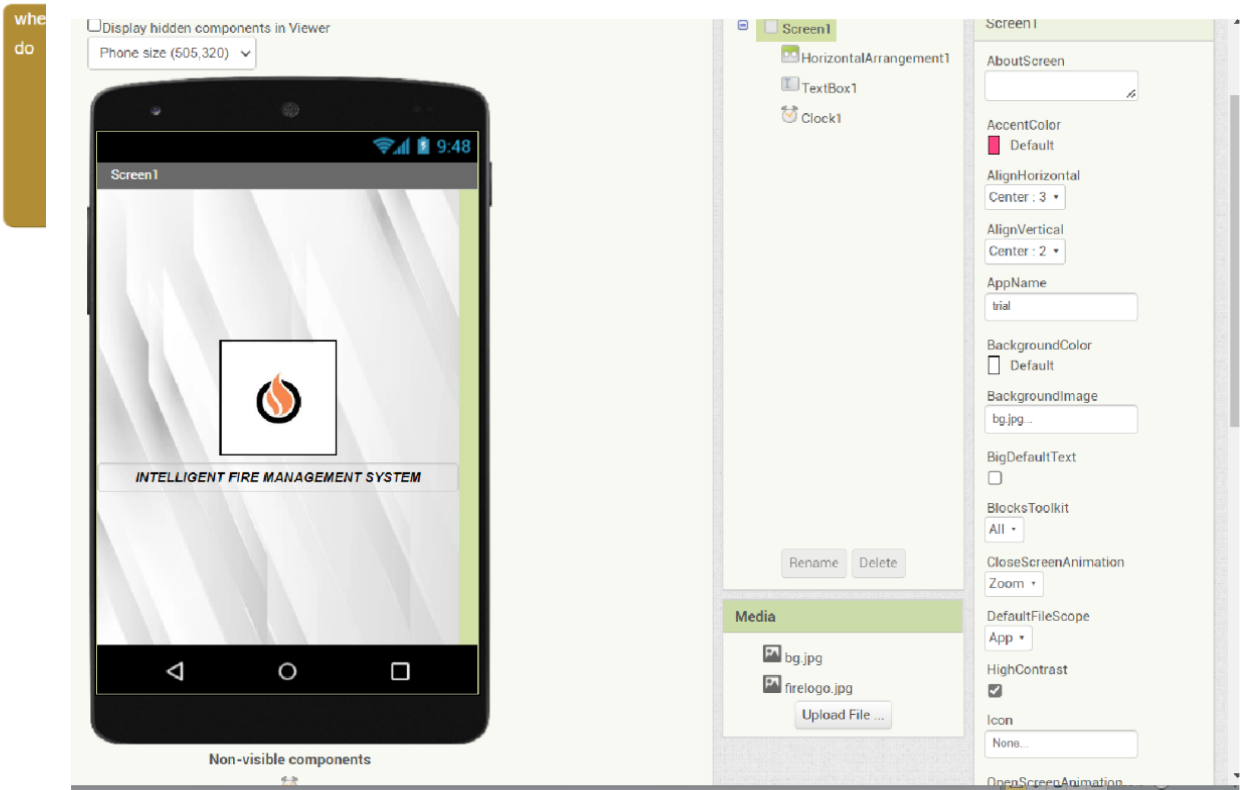
	id	key	value
<input type="checkbox"/>	657846f21e0cb8ead462fd89321d28...	657846f21e0cb8ead462fd89321d28...	{ "rev": "1-1c9683229f242d4133b7f..." }
<input type="checkbox"/>	657846f21e0cb8ead462fd89321dd3...	657846f21e0cb8ead462fd89321dd3...	{ "rev": "1-8aeec9d453a632f539ee9c..." }
<input type="checkbox"/>	657846f21e0cb8ead462fd8932201e...	657846f21e0cb8ead462fd8932201e...	{ "rev": "1-7b6df30912cf9fde43ca8b..." }
<input type="checkbox"/>	657846f21e0cb8ead462fd8932203d...	657846f21e0cb8ead462fd8932203d...	{ "rev": "1-a9bec25d7f94ccc71ce692..." }
<input type="checkbox"/>	70ea2e4bb2a9c635be3ce2603a25a...	70ea2e4bb2a9c635be3ce2603a25a...	{ "rev": "1-b567b4cce122c31e1666fc..." }
<input type="checkbox"/>	70ea2e4bb2a9c635be3ce2603a268...	70ea2e4bb2a9c635be3ce2603a268...	{ "rev": "1-217497b95c16c3d228800..." }
<input type="checkbox"/>	70ea2e4bb2a9c635be3ce2603a272...	70ea2e4bb2a9c635be3ce2603a272...	{ "rev": "1-a01738b27517a2bb4b93b..." }
<input type="checkbox"/>	70ea2e4bb2a9c635be3ce2603a273...	70ea2e4bb2a9c635be3ce2603a273...	{ "rev": "1-13230a9f364a021a02422..." }
<input type="checkbox"/>	7170def319e06e12e85b74c728897...	7170def319e06e12e85b74c728897...	{ "rev": "1-4dbfcfb4dbbf888784fc24d..." }
<input type="checkbox"/>	7170def319e06e12e85b74c7288b7...	7170def319e06e12e85b74c7288b7...	{ "rev": "1-5b1a46d23a6c259bd5b97..." }
<input type="checkbox"/>	7170def319e06e12e85b74c7288c2...	7170def319e06e12e85b74c7288c2...	{ "rev": "1-782ab5fa08aed27641a1..." }

Showing document 1 - 20. Documents per page: 20 ▾ < >

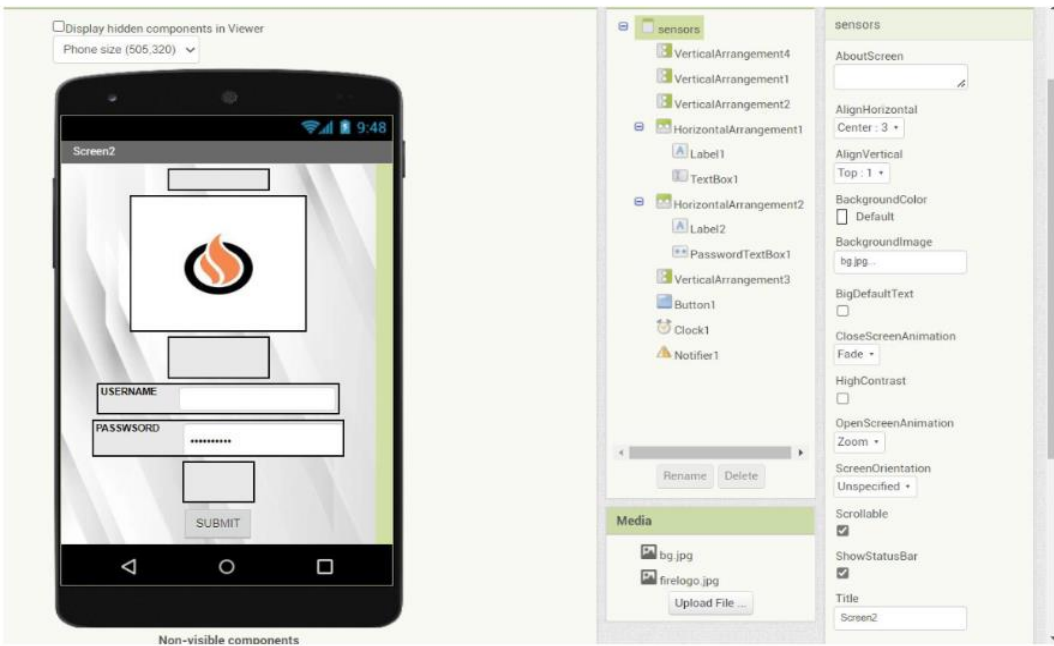


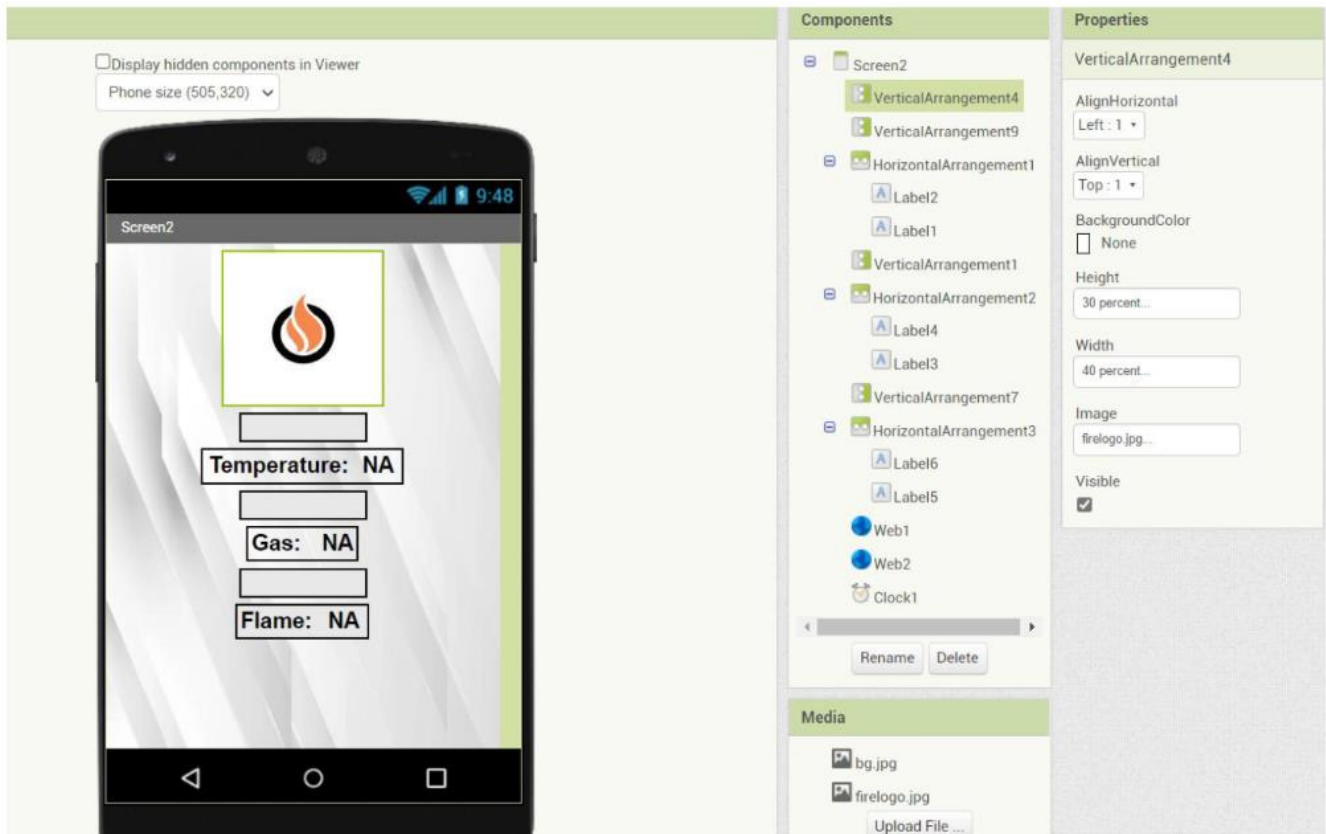
SPRINT – 4:

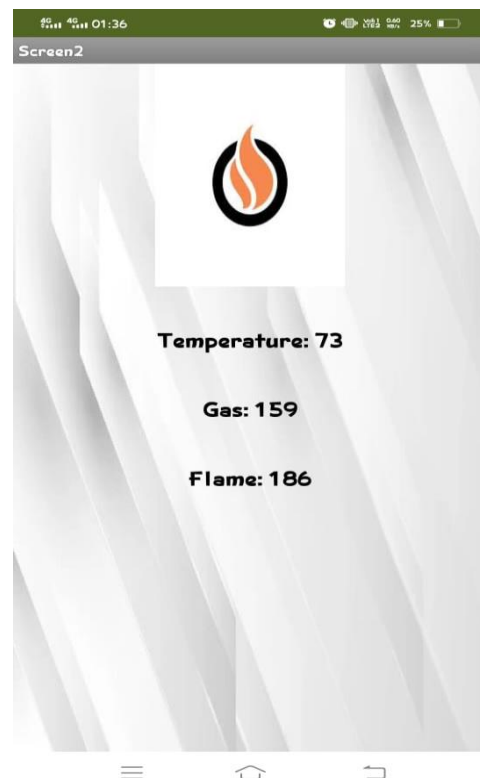
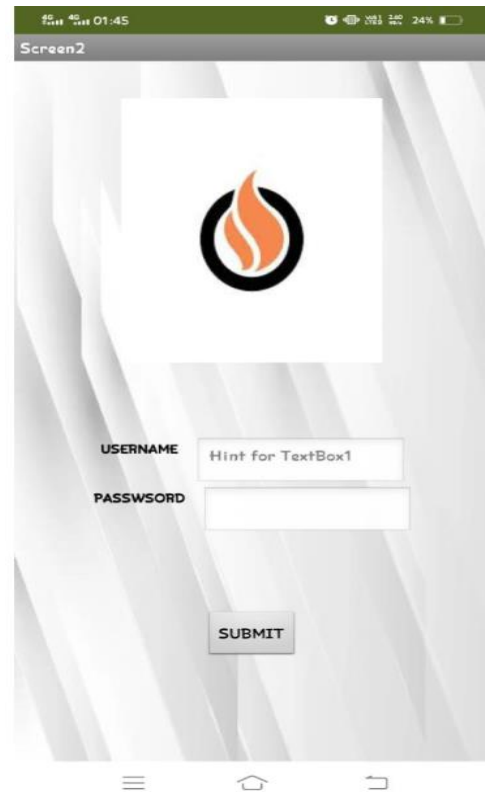
MIT APP INVENTOR:



```
when Clock1.Timer
do
  set Clock1.TimerEnabled to false
  open another screen screenName sensors
```







RESULT

CPU USAGE:

The micro version of c++ is make the best use of the CPU. For every loop the program runs in one time, neglecting the network and communication. The program sleeps for every 1 second for better communication with MQTT. As the program takes $O(1)$ time and the compiler optimizes the program during compilation there is less CPU load for each cycle. The upcoming instructions are on the stack memory, so they can be popped after execution.

MEMORY USAGE:

The sensor values, networking data are stored in sram of the ESP32. It's a lot of data because ESP32 has only limited amount of memory (520 KB). For each memory cycle the exact addresses are overwritten with new values to save memory and optimal execution of the program.

ADVANTAGES:

Active monitoring for gas leakage and fire breakout

Automatic alerting of admin as well as fire authorities using SMS

Automatically turning on/off sprinkler as well as exhaust fan

Authentication is required to turn on/off of sprinkler and exhaust fan as well as sending SMS alert manually

It automatically detect false fire breakout reducing unnecessary panic by using flow sensors we can confirm that the sprinkler system is working as it intended

All device status can be shown in a dashboard

Users can see the dashboard using a web application Disadvantages

Always need to connect with the internet [Only to Send the SMS alert]

If the physical device is damaged the entire operation is collapsed

Need large database since many data is stored in cloud database every second

CONCLUSION:

So, in conclusion our problem premise is solved using IoT devices by creating a smart management system that solves many inherent problems in the traditional fire management system like actively monitoring for fire breakouts as well as gas leakage and sending SMS alerts to the admin as well as to the fire authorities.

FUTURE SCOPE:

The existing devices can be modified to work in different specialized environment as well as scale to house use to big labs [Since fire accidents can cause major loss in human lives in homes to big industries] as well as it can be used in public places, vehicles.

APPENDIX

Esp32 - Microcontroller:

ESP32 is a series of low-cost, low-power system on a chip microcontroller with integrated Wi-Fi and dual-mode Bluetooth

Memory: 320 KiB SRAM

CPU: Tensilica Xtensa LX6 microprocessor @ 160 or 240 MHz

Power: 3.3 V DC

Manufacturer: Espressif Systems

Predecessor: ESP8266

SENSORS:

DHT22 - Temperature and Humidity sensor

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed).

24 MQ5 - Gas sensor

Gas sensors (also known as gas detectors) are electronic devices that detect and identify different types of gasses. They are commonly used to detect toxic or explosive gasses and measure gas concentration.

Flame sensors

A flame-sensor is one kind of detector which is mainly designed for detecting as well as responding to the occurrence of a fire or flame. The flame detection response can depend on its fitting

SOURCE CODE:

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

# Provide your IBM Watson Device Credentials
organization = "gh6uoi"
deviceType = "fire"
deviceId = "fire123"
authMethod = "token"
authToken = "0123456789"

# 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 ()


def myCommandCallback(cmd) :

    print ( "Command received: %s" % cmd.data['command'] )

    status = cmd.data['command']

def myCommandCallback(cmd) :

    print ( "Command received: %s" % cmd.data['command'] )

    status = cmd.data['command']

def myCommandCallback(cmd) :

    print ( "Command received: %s" % cmd.data['command'] )

    status = cmd.data['command']

while True :

    # Get Sensor Data from DHT11

    temp = random.randint ( 0, 100 )

    gas = random.randint ( 60, 200 )

    flame = random.randint ( 60, 200 )

    data = { 'temp' : temp, 'Gas' : gas, 'Flame': flame }

    # print data

    def myOnPublishCallback() :

        print ( "Published Temperature = %s C" % temp, "Gas = %s %" % gas, "Flame = %s %" % flame, "to IBM Watson")

        success = deviceCli.publishEvent ( "IoTSensor", "json", data, qos = 0, on_publish = myOnPublishCallback )

        if not success :

```

```
    print ( "Not connected to IoT" )  
time.sleep ( 1 )  
deviceCli.commandCallback = myCommandCallback  
# Initialize GPIO  
if temp > 50 :  
    print ( "buzzer is on" )  
else :  
    print ( "buzzer is off" )  
if flame > 100 :  
    print ( "sprinklers are on" )  
else :  
    print ( "sprinklers are off" )  
if gas>100:  
    print ( "exhaust fan is on" )  
else :  
    print ( "exhaust fan is off" )  
# Disconnect the device and application from the cloud  
deviceCli.disconnect ()
```

GITHUB PROJECT LINK:

<https://github.com/IBM-EPBL/IBM-Project-6983-1658844753>

PROJECT DEMO VIDEO LINK:

<https://drive.google.com/folderview?id=1LhTxg2hSZhHTa8VmnXbGn8idr4wbfx4a>