

**INDUSTRY-SPECIFIC INTELLIGENT FIRE MANAGEMENT SYSTEM**

**PROJECT REPORT**

|  |  |
| --- | --- |
| Team ID | PNT2022TMID42730 |
| Project Name | Industry-Specific Intelligent Fire Management System |

Submitted by :

Suryakala S **Team ID:**PNT2022TMID47230

Gowtham D **Industry Mentor** : Santoshi

Mohana sundaram A **Faculty Mentor** : S.K.Kabilesh

Rajesh L

**Title Page No**

1. [INTRODUCTION 3](#_TOC_250012)
   1. [Project Overview **3**](#_TOC_250011)
   2. [Purpose **3**](#_TOC_250010)
2. [LITERATURE SURVEY 3](#_TOC_250009)
   1. [Existing problem **4**](#_TOC_250008)
   2. [References **4**](#_TOC_250007)
   3. [Problem Statement Definition **4**](#_TOC_250006)
3. [IDEATION & PROPOSED SOLUTION 4](#_TOC_250005)
   1. [Empathy Map Canvas **4**](#_TOC_250004)
   2. [Ideation & Brainstorming **5**](#_TOC_250003)
   3. [Proposed Solution **6**](#_TOC_250002)
   4. [Problem Solution fit **7**](#_TOC_250001)
4. REQUIREMENT ANALYSIS 8
   1. Functional requirement **8**
   2. Non-Functional requirements **9**
5. PROJECT DESIGN 10
   1. Data Flow Diagrams **10**
   2. Solution & Technical Architecture **11**
   3. User Stories **13**
6. PROJECT PLANNING & SCHEDULING 14
   1. Sprint Planning & Estimation **14**
   2. Sprint Delivery Schedule **14**
   3. Reports from JIRA **15**
7. CODING & SOLUTIONING 17
   1. Feature 1 **17**
   2. Feature 2 **19**
   3. Feature 3 **21**
8. TESTING 23
   1. Test Cases **23**
   2. User Acceptance Testing **24**
9. RESULTS 25
   1. Performance Metrics **25**
10. ADVANTAGES & DISADVANTAGES 27
11. CONCLUSION 27
12. FUTURE SCOPE 28
13. APPENDIX 28

[Source Code **31**](#_TOC_250000)

# INTRODUCTION

## Project Overview

The "Industry specific-Intelligent fire management system's" goal is to prevent unintentional fire accidents in industries and to take the necessary precautions to prevent any mishaps. A Gas sensor, Flame sensor, and Temperature sensor are all part of the smart fire management system to monitor environmental changes. The sprinklers will be turned on automatically if any flame is found. 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 readings and if any Gases are present, the exhaust fans are turned ON. These readings are continuously tracked by IBM Watson IOT Platform and saved in Cloudant DB. Through the Nexmo SMS API, the police and fire station will be informed if any variations take place. Authorities and the fire station are informed of emergency notifications.

## Purpose:

* + - To provide an easy management system on the dashboard .
    - Providing an overview of the user's experience.
    - The ability to use IoT devices to detect the status of a room
    - To turn on sprinklers and exhaust fans in the event of an accident.
    - To send and store temperature status in cloud storage.
    - To send an SMS to the authorities in the event of a fire accident.

# LITERATURE SURVEY

## Existing problem:

The lack of a dependable, effective, cost-effective, modern processing, or feature-rich fire management system in many buildings, as well as the fact that it lacks an automatic alarm system for administrators and authorities, make the situation less than ideal. The sprinkler system cannot even be activated since they are utilising outdated fire safety technologies, and none of them effectively interact with one another to prevent false alarms. Applications are also being used to monitor the entire system.

## References:

[https://pdfs.semanticscholar.org/f3e7/a7c0cf2d448be592](https://pdfs.semanticscholar.org/f3e7/a7c0cf2d448be592421045033506e845e6c2.pdf) [421045033506e845e6c2.pdf](https://pdfs.semanticscholar.org/f3e7/a7c0cf2d448be592421045033506e845e6c2.pdf)

[**https://www.mdpi.com/2224-2708/7/1/11**](https://www.mdpi.com/2224-2708/7/1/11)

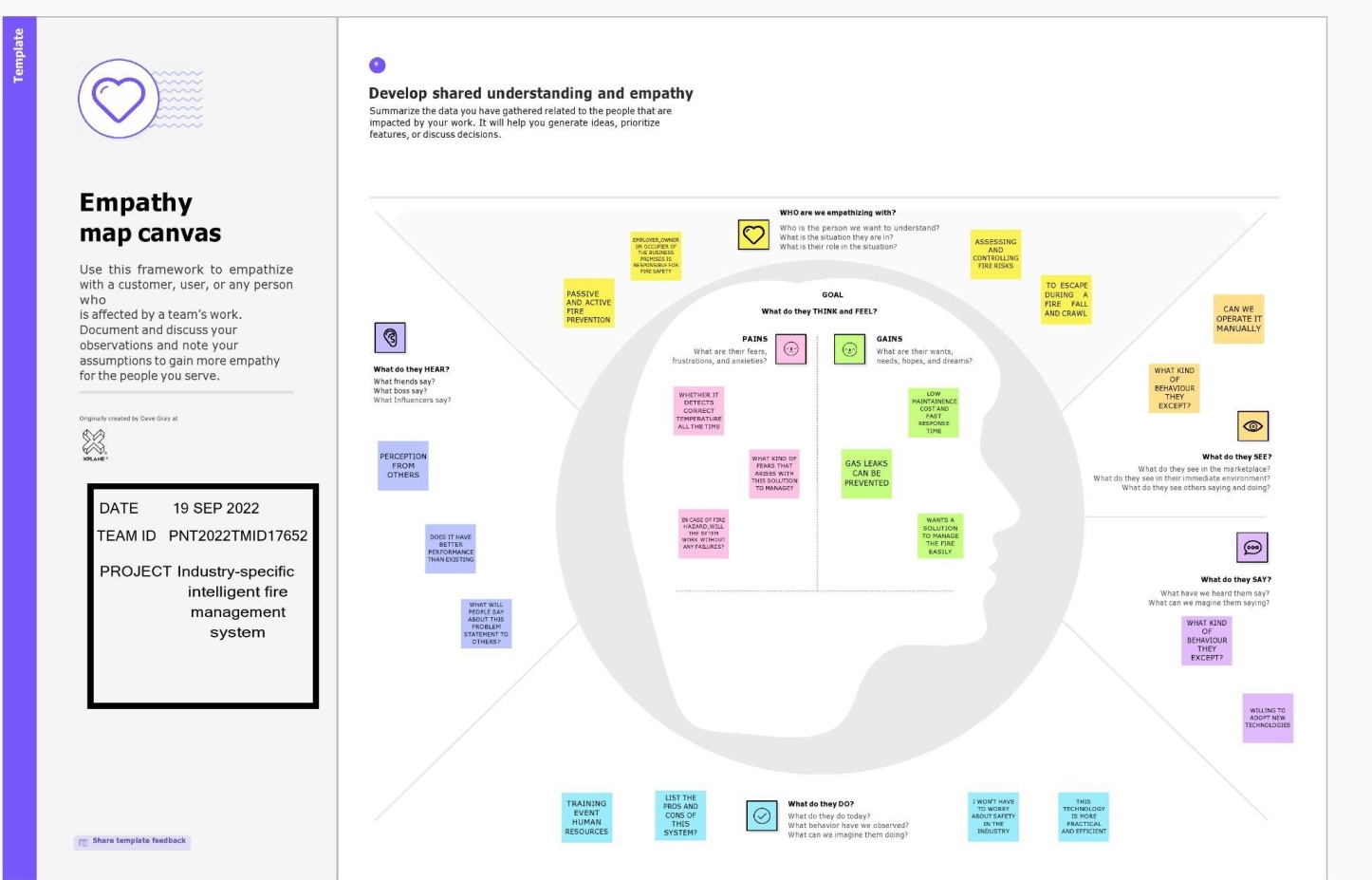
## Problem Statement Definition:

The fire management systems in homes and businesses are not very dependable, efficient, or affordable, and they lack features like an automatic alert system for administrators and authorities. Many buildings still use outdated fire safety systems that can't even activate the sprinkler system, and they all improperly communicate with one another to prevent false alarms. They also use applications to monitor the entire system.

## IDEATION & PROPOSED SOLUTION:

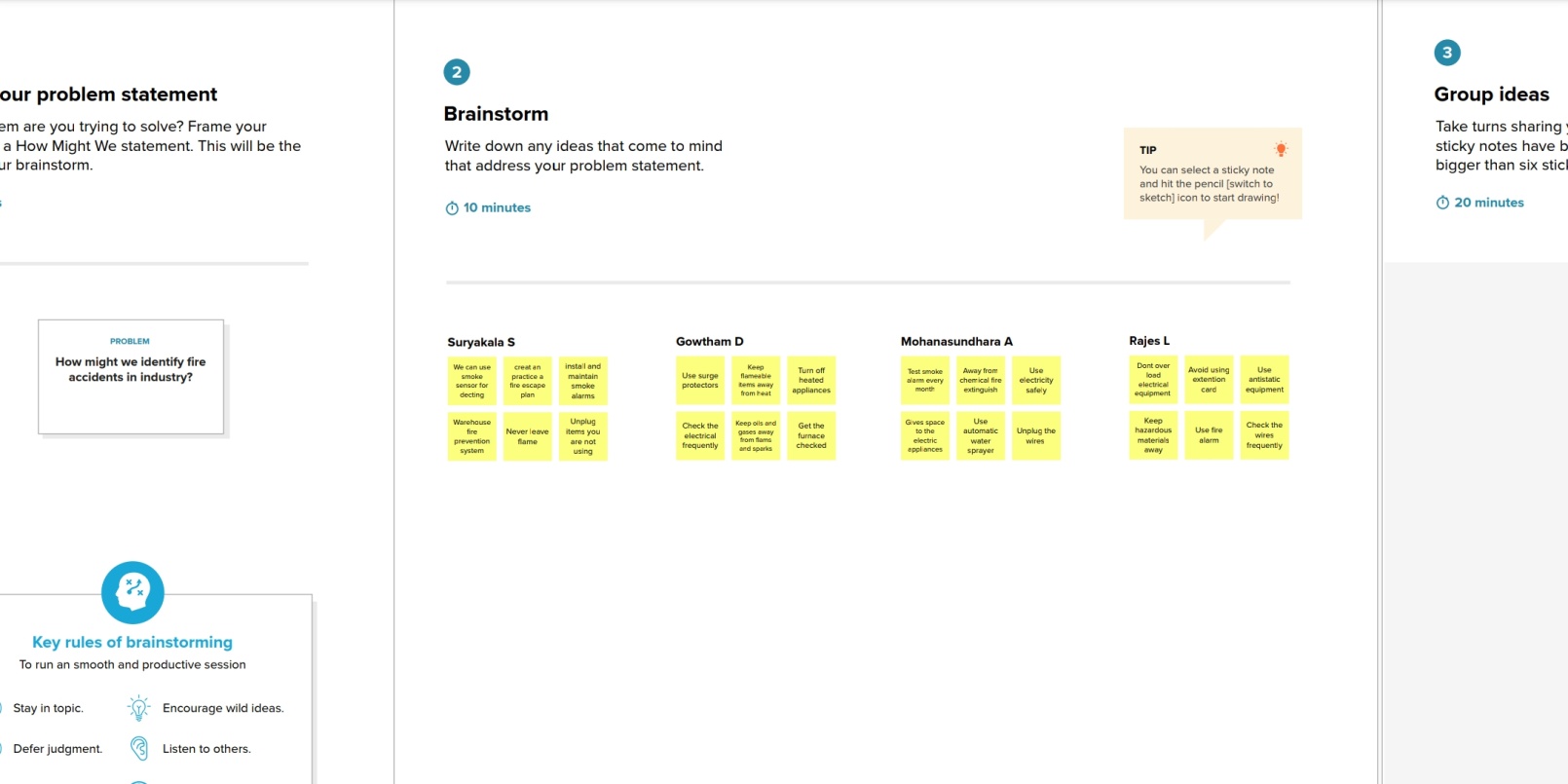
### Empathy Map Canvas:

* + - An empathy map is a straightforward, simple-to-understand picture that summarises information about a user's actions and views.
    - It is a helpful tool that enables teams to comprehend their users more fully. It's important to comprehend both the actual issue and the individual who is experiencing it in order to develop a workable solution.
    - Participants learn to think about situations from the user's perspective, including goals and problems, through the exercise of constructing the map.

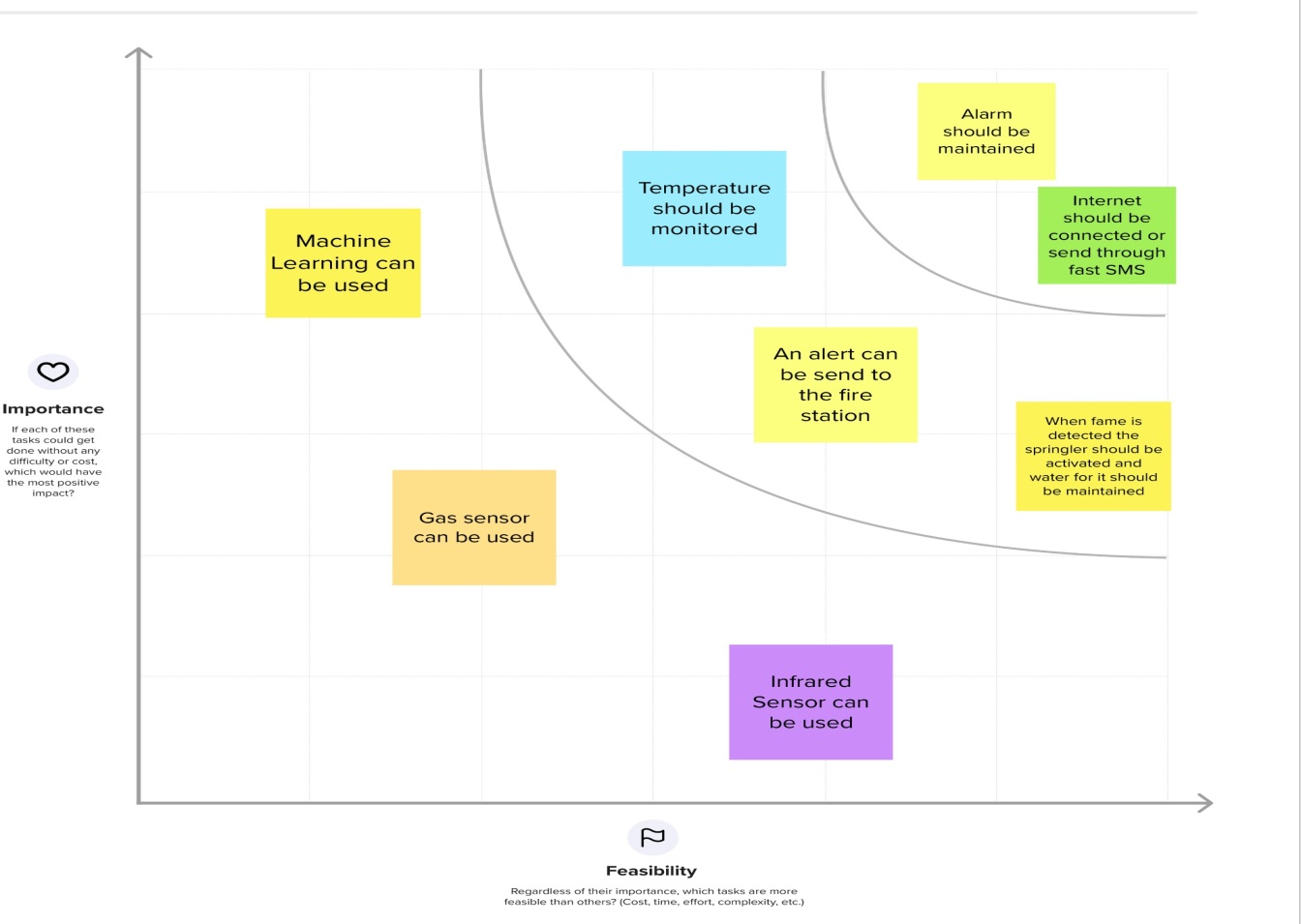


* 1. **Ideation & Brainstorming**

**Step 1:Brainstorm, Idea Listing and Grouping:**



## Step 2:Idea Prioritisation:

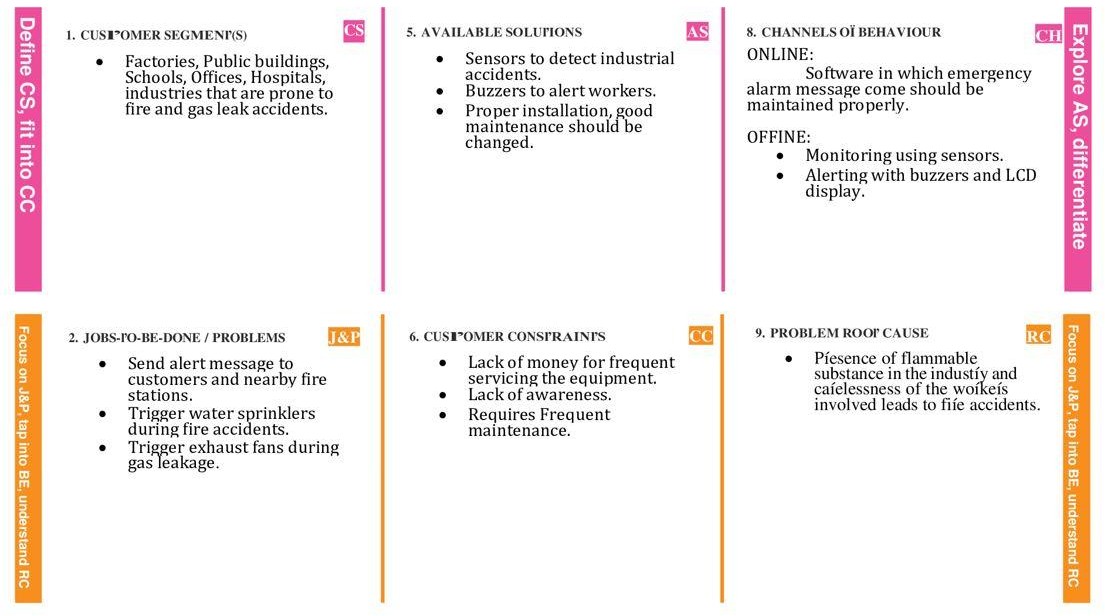


* 1. **Proposed Solution:**

|  |  |  |
| --- | --- | --- |
| **S.N**  **o.** | **Parameter** | **Description** |
| 1. | Problem Statement (Problem to be solved) | To address this problem, this aims to implement a smart fire detection system that would not only detect the fire using integrated sensors but also alert property owners, emergency services, and local police stations to protect lives and valuable assets simultaneously. |

|  |  |  |
| --- | --- | --- |
| 2. | Idea / Solution description | The proposed model in this problem statement employs different integrated detectors, such as heat, smoke, and flame. The signals from those detectors go through the system algorithm to check the fire's potentiality and then broadcast the predicted result to various parties using GSM modem associated with the system. Finally, the main feature of the proposed system is to minimise false alarms,  which, in turn, makes this system more reliable. |
| 3. | Novelty / Uniqueness | To get real-life data without putting human lives in danger, an IoT technology has been implemented to provide the fire department with the necessary data. |
| 4. | Social Impact / Customer Satisfaction | * Highly accurate. * It prevents accidents caused by fire in industries. * No need for manpower. * Human risk is low. |
| 5. | Business Model (Revenue Model) | * High Secure. * Our model will help industries by preventing huge losses that occur due to fire accidents. |
| 6. | Scalability of the Solution | Since our model is cost effective because of usage of multiple sensors any and every kind of industry can use our Industry Specific Intelligent Fire Management System and it produces least false alarms. |

* 1. **Problem Solution fit:**



# REQUIREMENT ANALYSIS

## Functional requirement:

* + - A system's or component's function is defined by a functional requirement, where a function is defined as the behaviour between inputs and outputs.
    - It specifies what the software system “ should do” ?
    - It is defined at the component level and aids in software functionality verification.

|  |  |  |
| --- | --- | --- |
| **FR**  **No.** | **Functional Requirement (Epic)** | **Sub Requirement (Story / Sub-Task)** |
| FR-1 | Emergency alert | Alert through SMS.  Alert through audible and visible alarms |
| FR-2 | User Understanding | Based on the data, the user understands that if any of the data is above the threshold value, then there is a  fire burst. |
| FR-3 | User action | In case of fire bursts, the user needs to take actions like find the best escape route, evacuate the workers and  take necessary actions to control the fire. |

|  |  |  |
| --- | --- | --- |
| FR-4 | Control functions | Activation of duct mounted smoke mounted detector will shut down the heating ventilation and air conditioning equipment to prevent the migration of  smoke to non-affected areas of the building |
| FR-5 | Location notification | Location of fire must be sent to fire department through an alarm. |

## Non-Functional requirements:

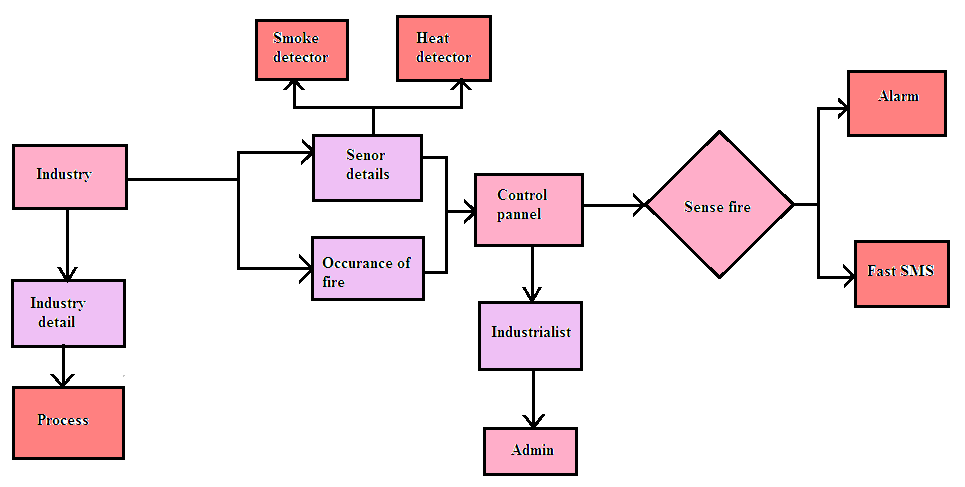
* + - A software system's quality characteristic is defined by a non-functional need.
    - The question of "How should the software system fulfil the functional requirements?" is constrained.
    - Assists you in evaluating the software's performance

|  |  |  |
| --- | --- | --- |
| **FR**  **No.** | **Non-Functional Requirement** | **Description** |
| NFR- 1 | Usability | * Visual and audio signalization. * It provides zonal coverage. * Protect your property |
| NFR- 2 | Security | * Warn people when smoke ,fire,carbon monoxide. * Ensure the protection of both valuable items and human life. |
| NFR- 3 | Reliability | * Response timer will be faster * Reliable fire alarm systems are largely influenced. * It may be capable of precisely identifying the smoke, and it doesn't issue an erroneous warning or signal. |

|  |  |  |
| --- | --- | --- |
| NFR- 4 | Performance | 1. Detect a fire. 2. Alert occupants of the fire condition. 3. Activate safety control functions.   Alert the local fire department. |
| NFR- 5 | Availability | * Ability to use the system for other types of emergency communication. * It is useful to people because it is accessible throughout the day and night. |
| NFR- 6 | Scalability | The sensors and boards used in this system should be able to easily alter and overhaul in accordance with  required changes. |

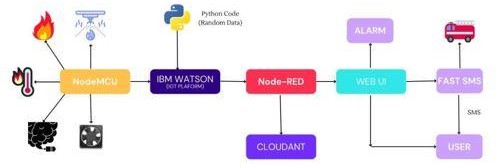
# PROJECT DESIGN:

## Data Flow Diagrams:

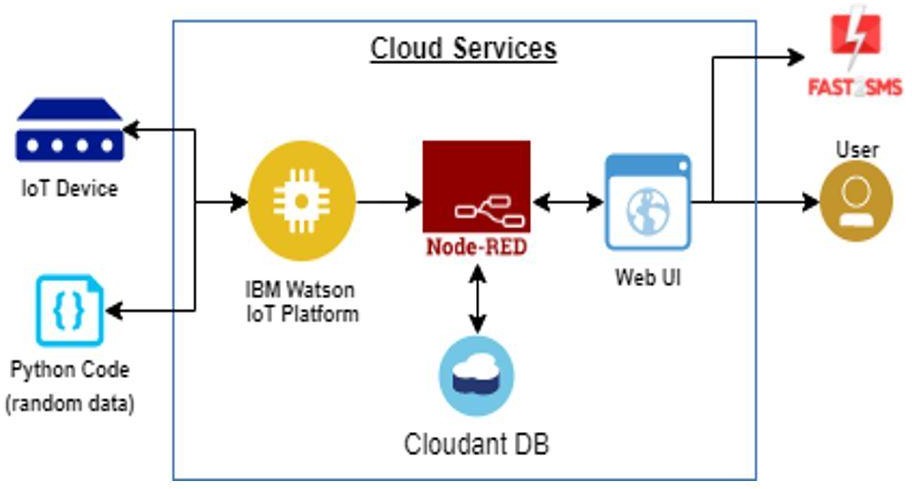


* 1. **Solution & Technical Architecture:**

**Solution Architecture:**

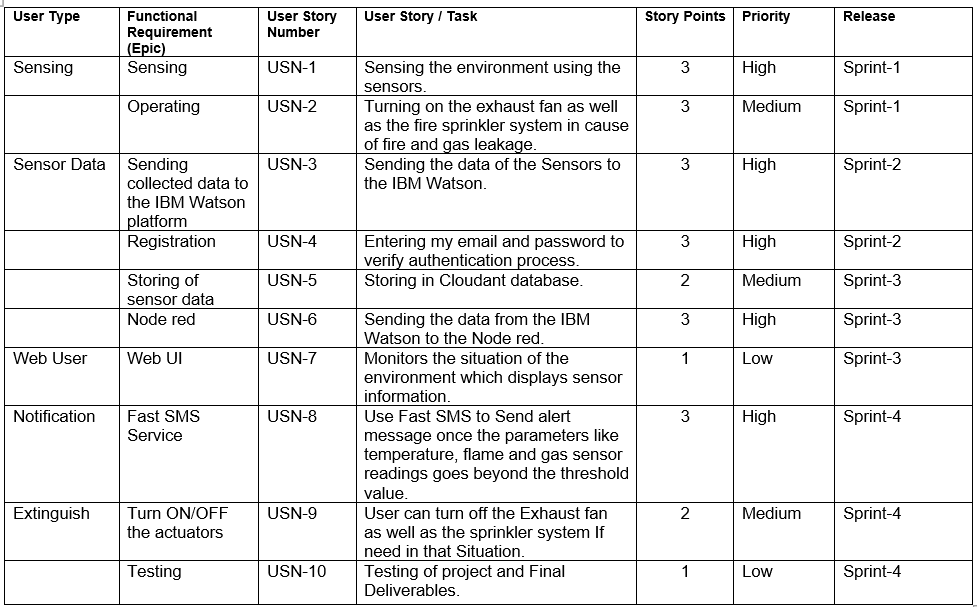


**Technical Architecture:**



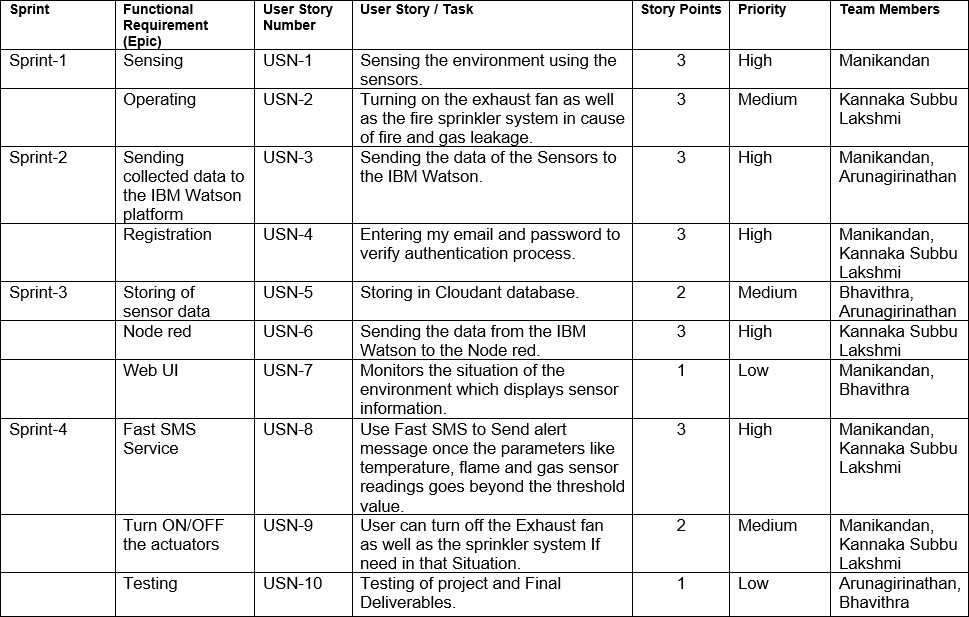
|  |  |  |  |
| --- | --- | --- | --- |
| **S.**  **No** | **Component** | **Description** | **Technology** |
| 1  . | User Interface | Web UI, Node-RED, MIT app | IBM IoT Platform, IBM Node red, IBM Cloud |
| 2  . | Application Logic-1 | Create Ibm Watson IoT platform and create node- red service | IBM Watson, IBM cloud ant service, IBM node-red |
| 3  . | Application Logic-2 | Develop python script to publish and subscribe to IBM IoT Platform | python |
| 4  . | Application Logic-3 | Build a web application using node-red service | IBM Node-red |
| 5  . | Database | Data Type, Configurations etc. | MySQL |
| 6  . | Cloud Database | Database Service on Cloud | IBM DB2, IBM  Cloudant |
| 7  . | File Storage | Developing mobile application to store and receive the sensors information and to react accordingly | Web UI, python |
| 8  . | External API-1 | Using this IBM fire management API, we can track the temperature of the incident place and where the fire had been attacked. | IBM fire management API |
| 9  . | External API-2 | Using this IBM Sensors it detects the fire, gas leaks, temperature and provides the activation of sprinklers to web UI | IBM Sensors |
| 10. | Machine Learning Model | Using this we can derive the object recognition model | Object Recognition Model |
| 11. | Infrastructure (Server  / Cloud) | Application Deployment on Local System / Cloud Server Configuration | IBM cloud ant, IBM IoT Platform |

* 1. **User stories:**

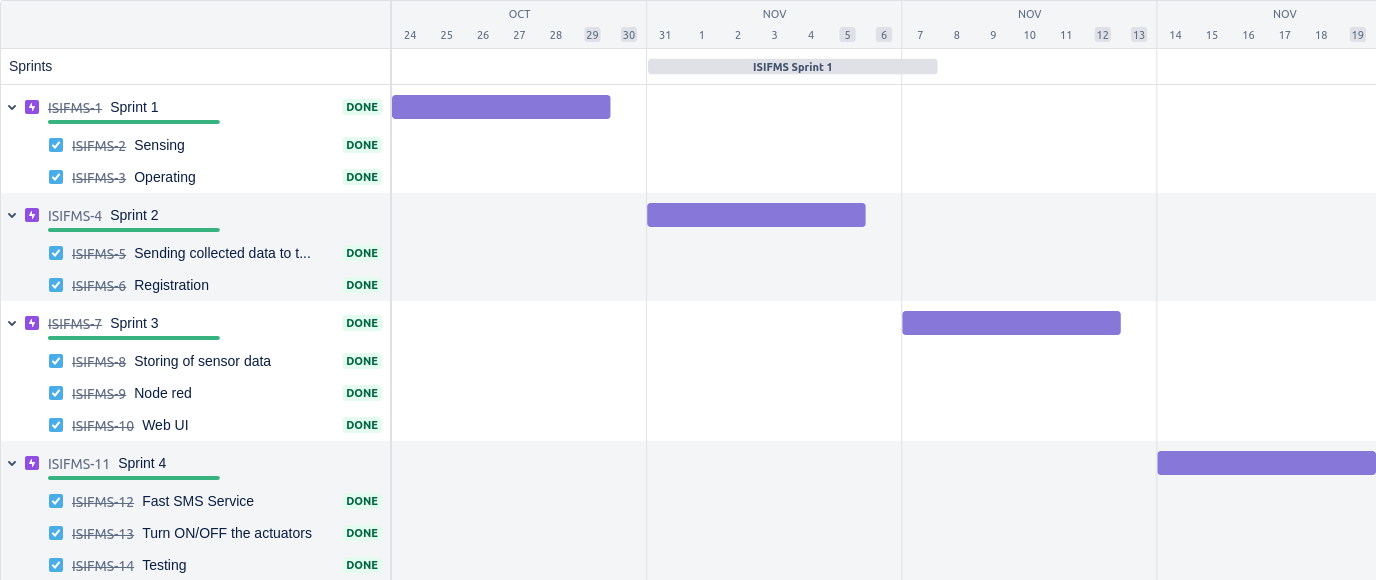


# PROJECT DESIGNING AND PLANNING:

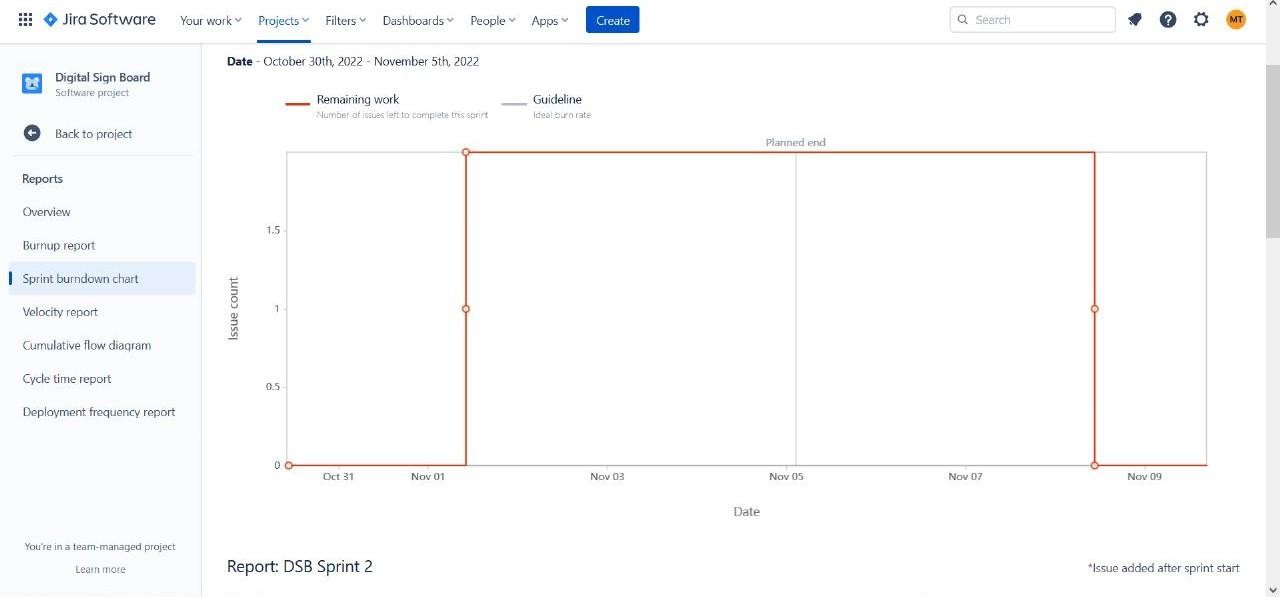
## Sprint planning and estimation



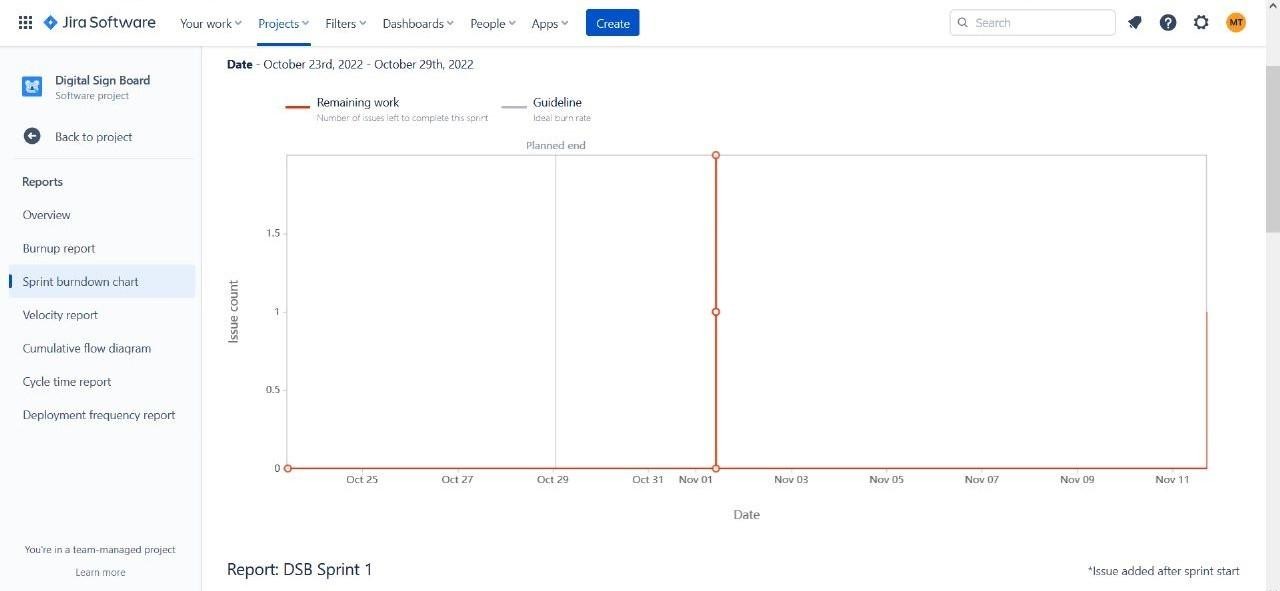
## Reports from JIRA:



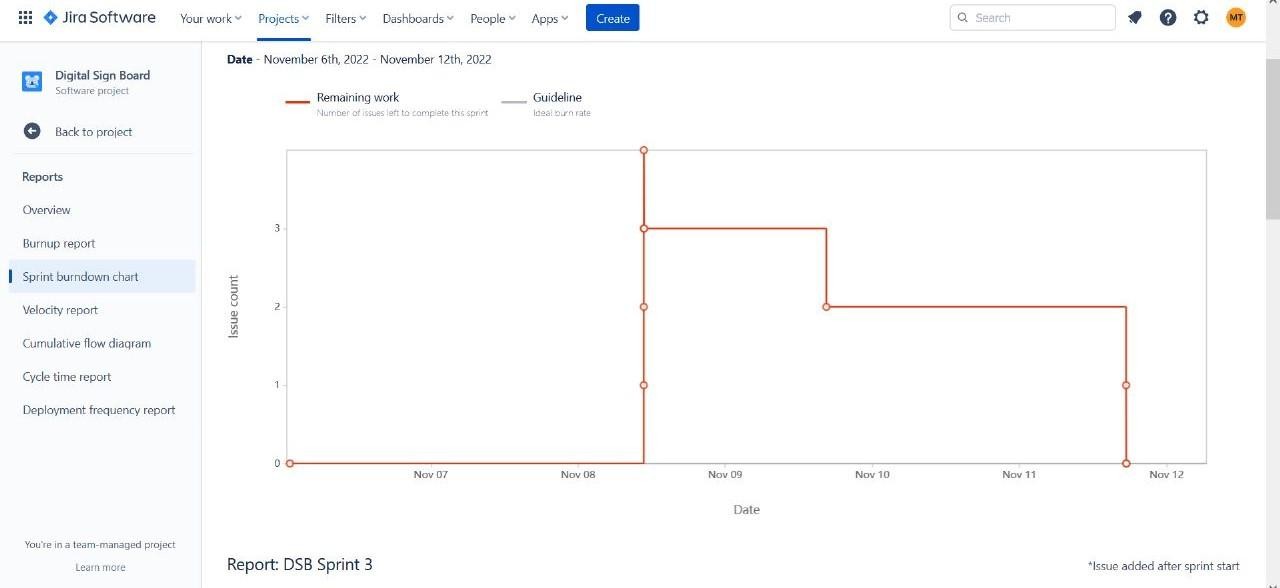
**Sprint 1:**



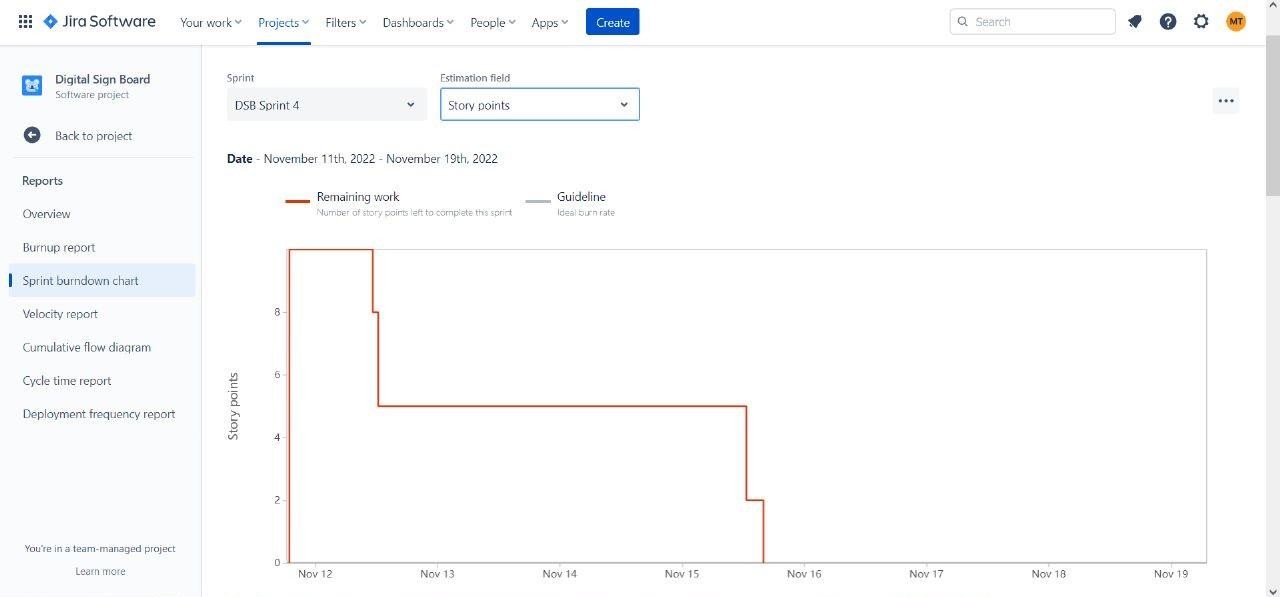
## Sprint 2



### Sprint 3



**Sprint 4**



# CODING AND SOLUTIONING:

## Feature 1 : alarm checking

### //initial variable

**temperature = random(-20,125); gas = random(0,1000);**

### int flamereading = random(200,1024); flame = map(flamereading,0,1024,0,2);

**//set a flame status**

### switch (flame) { case 0:

**flame\_status = "No Fire"; Serial.println("Flame Status :**

### "+flame\_status);

**break; case 1:**

### flame\_status = "Fire is Detected"; Serial.println("Flame Status :

**"+flame\_status);**

### break;

**}**

### //Gas Detection if(gas > 100){

**Serial.println("Gas Status : Gas leakage Detected");**

### }

**else{**

### exhaust\_fan\_on = false;

**Serial.println("Gas Status : No Gas leakage Detected");**

### }

**//send the sprinkler status if(flame){**

### sprinkler\_status = "working"; Serial.println("Sprinkler Status :

**"+sprinkler\_status);**

### }

**else{**

### sprinkler\_status = "not working"; Serial.println("Sprinkler Status :

**"+sprinkler\_status);**

### }

**//toggle the fan according to gas if(gas > 100){**

### exhaust\_fan\_on = true; Serial.println("Exhaust fan Status :

**Working");**

### }

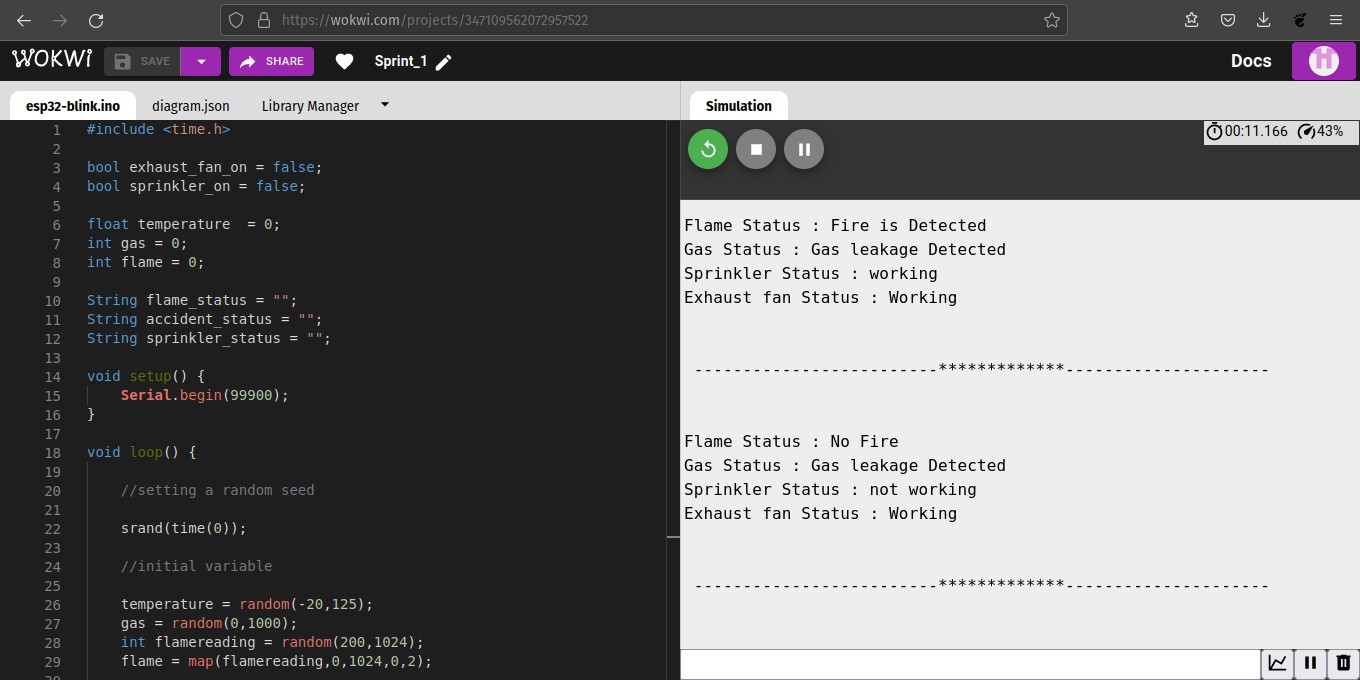
**else{**

### exhaust\_fan\_on = false; Serial.println("Exhaust fan Status : Not

**Working");**

**}**

## Result:



**Explanation**

* This set of code checks for false alarms.
* It also gives the current status of actuators.

## Feature 2 : Sending data into IBM Watson (JSON)

### String payload = "{";

**payload+="\"gas\":"; payload+=gas; payload+=",";**

### payload+="\"temperature\":"; payload+=(int)temperature; payload+=","; payload+="\"flame\":"; payload+=flamereading; payload+=",";

**payload+="\"fire\_status\":\""+flame\_status+"\",";**

### payload+="\"sprinkler\_status\":\""+sprinkler\_status+"\ ",";

**payload+="\"Gas\_status\":\""+Gas\_status+"\",";**

### payload+="\"exhaust\_fan\_status\":\""+exhaust\_fan\_statu s+"\"}";

**if(client.publish(publishTopic, (char\*) payload.c\_str()))**

### {

**Serial.println("Publish OK");**

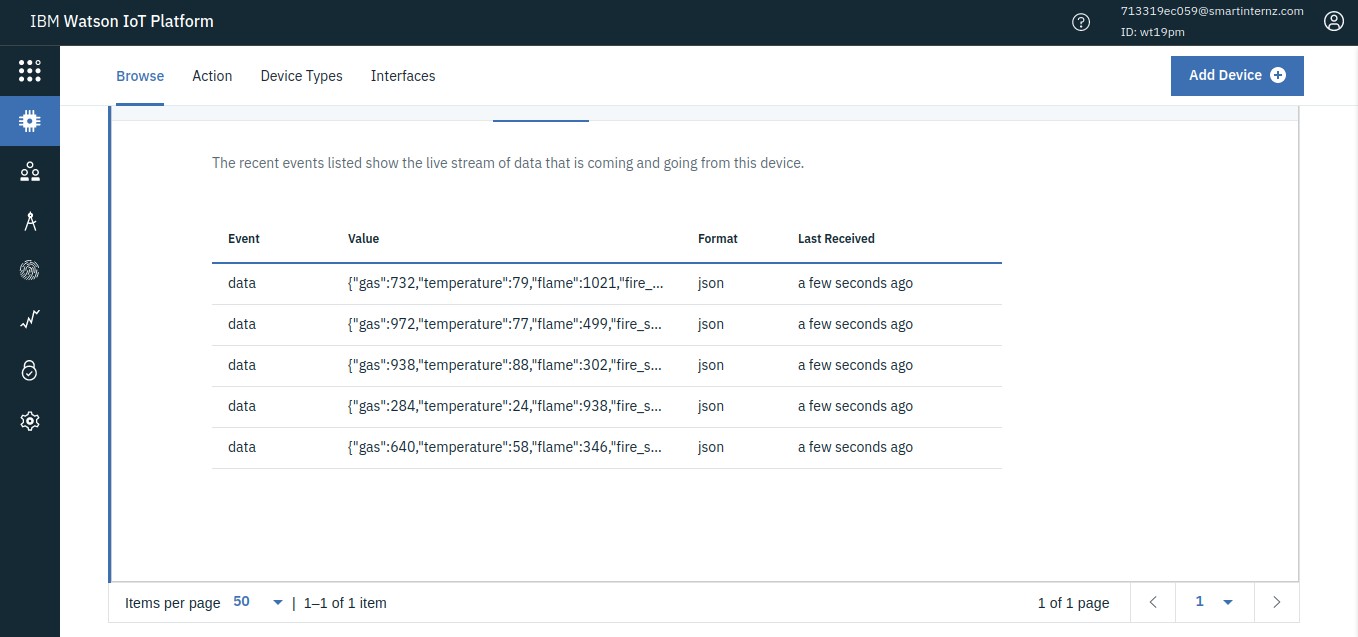
### }

**else{**

### Serial.println("Publish failed");

**}**

## Result:



**Explanation:**

* It sends the data to IBM IoT Watson platform.

## Feature 3 :

### //handles commands from user side

**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];**

### }

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

### const char \*s =(char\*) data3.c\_str(); double pincode = 0;

**if(mjson\_get\_number(s, strlen(s), "$.pin", &pincode)){**

### if(((int)pincode)==67993){ const char \*buf;

**int len;**

### if (mjson\_find(s, strlen(s), "$.command", &buf, &len)) // And print it

**{**

### String command(buf,len); if(command=="\"cantfan\""){

**//this works when there is gas sensor reads high value and if there should be a**

### //manual trigger else it will be automate canfanoperate = !canfanoperate;

**}**

### else if(command=="\"cantsprink\""){ cansprinkoperate = !cansprinkoperate;

**}else if(command=="\"sentalert\""){**

### //this works when there is accident status is severe and if there should be a

**//manual trigger else it will be automate resetcooldown();**

### }

**}**

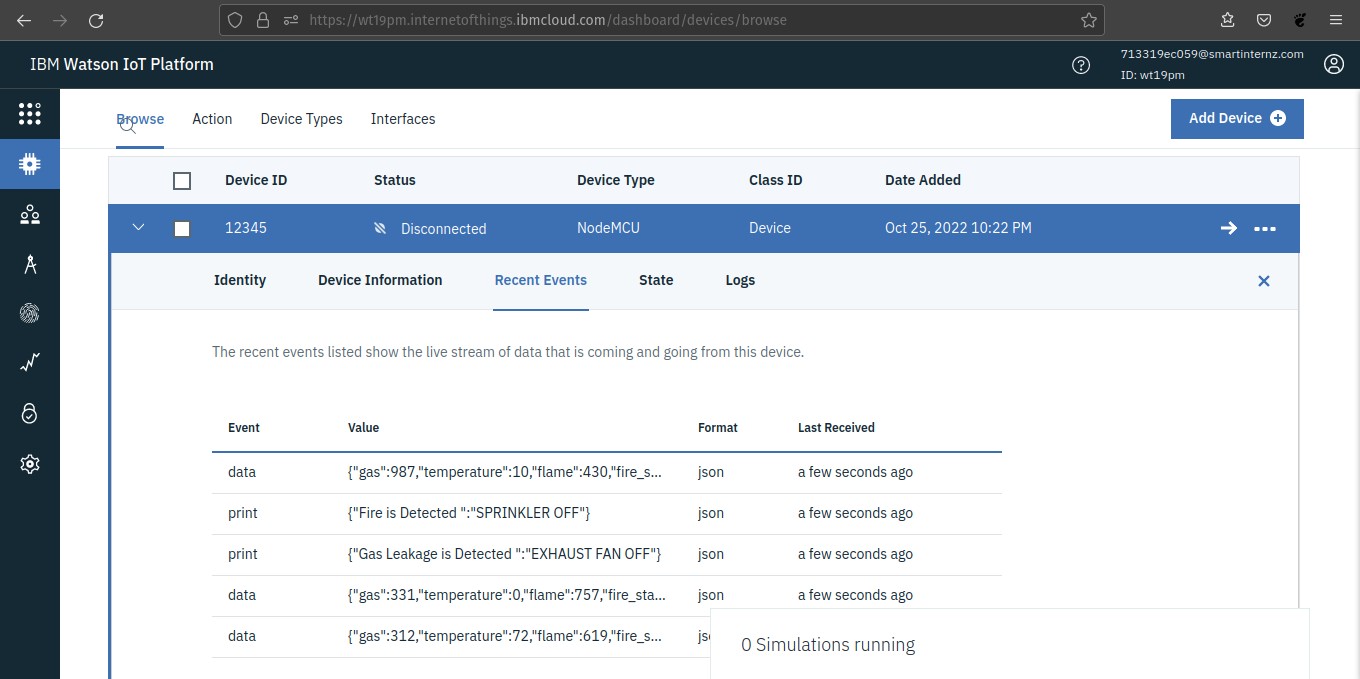
### }

**}**

### data3="";

**}**

## Result:

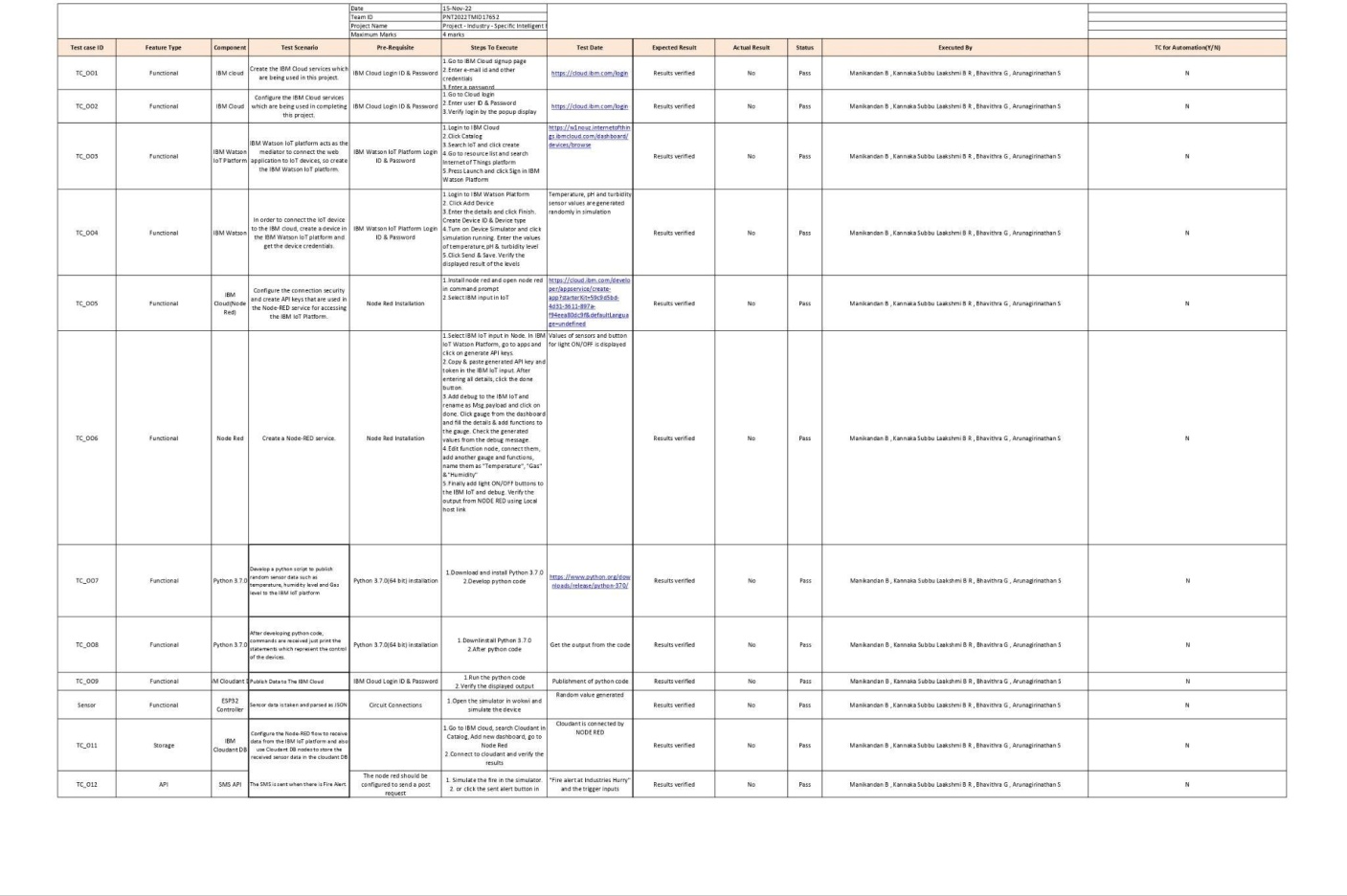


**Explanation:**

* The action taken by the user is received as a command and stored in a buffer.
* The event in the device is done according to the command.
* It checks for a secret encrypted pin for performing that event.

# TESTING:

* 1. **Testcases**



# UAT:

## Defect analysis:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

**Test case analysis:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Section** | **Total Cases** | **Not Tested** | **Fail** | **Pass** |
| Print Engine | 7 | 0 | 0 | 7 |
| Client Application | 51 | 0 | 0 | 51 |
| Security | 2 | 0 | 0 | 2 |
| Outsource Shipping | 3 | 0 | 0 | 3 |
| Exception Reporting | 9 | 0 | 0 | 9 |
| Final Report Output | 4 | 0 | 0 | 4 |
| Version Control | 2 | 0 | 0 | 2 |

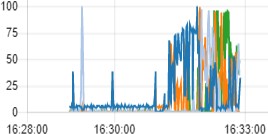
# RESULTS:

## performance metrics CPU usage

* + - Watson employs a cluster of ninety IBM Power 750 servers, each of which uses a 3.5 GHz POWER7 eight-core processor, with four threads per core. In total, the system has 2,880 POWER7 processor threads and 16

terabytes of RAM. According to John Rennie, Watson can process 500 gigabytes per second.





## Memory usage

* + - The sensor values , networking data are stored in sram of the ESP32 . It’s a lot of data because ESP32 has only a 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.

## Error rates

* + - The exceptions are handled in a proper way as it does not affect the usability of the system**.**
    - The errors rates are very low as the backend and dashboard is handled with node-red.

## Latency and Response Time

* + - For the data sent from the IoT device (considering the sleep of one second from the IoT ), the response is much quicker .We can easily see the delay caused by the sleep function The average time is well over optimal value.
    - Average time = (5𝑚𝑠 + 2600𝑚𝑠)/2 = 1302.5

## Garbage collection

* + - But it is not necessary in this scenario as the memory is used again for storing the data . Any dangling pointer or poorly handled address space is not allocated.

## ADVANTAGES AND DISADVANTAGES:

**Advantages**

* Checking constantly for gas leaks and fire starts.
* SMS-based automatic notification of administrative and fire authorities. Turning the exhaust fan and sprinklers on and off automatically.
* Sprinkler and exhaust fan operation, as well as manually sending SMS alerts, require authentication.
* It immediately detects erroneous fire breakout, which lessens needless fright. We may verify that the sprinkler system is operating as intended by employing flow sensors.
* A dashboard is capable of displaying all device status.

## Disadvantages

* To send the SMS alert, constant internet connection is required
* The entire operation falls apart if the physical apparatus is broken.
* A huge database is required since the cloud database stores a lot of data every second.

## CONCLUSION

* So, to sum up, our problem premise is resolved using IoT devices by developing a smart management system that addresses many inherent issues in the conventional fire management system.
* For example, the system actively monitors for fire breakouts as well as gas leakage and sends SMS alerts to the admin as well as the fire authorities.
* The live value is shown in the dashboard when this circuit uses a temperature, flame, and gas sensor.

## FUTURE SCOPE:

Since fire mishaps can result in significant loss of human life in both homes and large companies, the existing devices can be upgraded to operate in a variety of specialised environments and scaled for use in both public spaces and automobiles. In the event of any fire accidents, the police and fire station are notified.

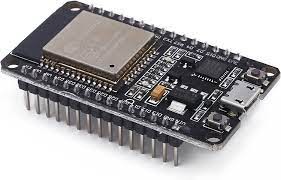
## APPENDIX:

**Esp32 - Microcontroller**

ESP32 is a series of low-cost, low-power [system on a chip](https://en.wikipedia.org/wiki/System_on_a_chip) [microcontrollers](https://en.wikipedia.org/wiki/Microcontroller) with integrated [Wi-Fi](https://en.wikipedia.org/wiki/Wi-Fi) and dual-mode [Bluetooth](https://en.wikipedia.org/wiki/Bluetooth).ESP32 is created and developed by [Espressif Systems](https://en.wikipedia.org/w/index.php?title=Espressif_Systems&action=edit&redlink=1), a Shanghai-based Chinese company, and is manufactured by [TSMC](https://en.wikipedia.org/wiki/TSMC) using their 40 nm process.

### Features:

* **Memory**: 320 KiB RAM, 448 KiB ROM
* **Wireless connectivity:**
  + Wi-Fi: [802.11](https://en.wikipedia.org/wiki/IEEE_802.11) b/g/n
  + Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi)
* **Peripheral interfaces:**
  + 34 × programmable [GPIOs](https://en.wikipedia.org/wiki/GPIO)
  + 12-bit [SAR ADC](https://en.wikipedia.org/wiki/Successive_approximation_ADC) up to 18 channels



## Sensors:

**DHT22 - Temperature 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). It's fairly simple to use but requires careful timing to grab data.

**Technical Detail:**

* + - Low cost
    - 3 to 5V power and I/O
    - 2.5mA max current use during conversion (while requesting data)
    - Good for 0-100% humidity readings with 2-5% accuracy



## MQ5 - Gas sensor:

MQ-5 gas sensor has high sensitivity to butane, propane, methane and can detect methane and propane at the same time. It also can detect kinds of flammable gases, especially LPG(propane). It is a kind of low–cost sensor for many applications.



## Flow Sensors:

A flow sensor is an electronic device that measures or regulates the flow rate of liquids and gases within pipes and tubes. Flow sensors are generally connected to gauges to render their measurements. Flow sensors are able to detect leaks, blockages, pipe bursts, and changes in liquid concentration.Flow sensors are of two groups:contact and non-contact flow sensors.



## Flame sensors:

A flame-sensor is one [kind of detector](https://www.elprocus.com/emf-detector-circuit-working-types-and-its-applications/) which is mainly designed for detecting as well as responding to the occurrence of a fire or flame. It includes an [alarm system](https://www.elprocus.com/fire-alarm-circuit-using-thermistor/), a natural gas line, propane & a fire suppression system. This sensor is used in [industrial boilers](https://www.elprocus.com/what-is-steam-boiler-working-principle-types-of-steam-boilers/). The main function of this is to give authentication whether the boiler is properly working or not. The response of these sensors is faster as well as more accurate compared with a

heat/smoke detector because of its mechanism while detecting the flame.



## Source code:

### #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 = "";

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

### wifiConnect(); mqttConnect();

**}**

### void loop() {

**srand(time(0));**

### //initial variables and random generated data

**temperature = random(-20,125); gas = random(0,1000);**

### int flamereading = random(200,1024); flame = map(flamereading,200,1024,0,2);

**//set a flame status**

### switch (flame) { case 0:

**flame\_status = "No Fire"; break;**

### case 1:

**flame\_status = "Fire is Detected"; break;**

### }

**//send the sprinkler status**

### if(flame==1){

**sprinkler\_status = "Working";**

### }

**else{**

### sprinkler\_status = "Not Working";

**}**

### //toggle the fan according to gas reading

**if(gas > 100){**

### Gas\_status = "Gas Leakage is Detected"; exhaust\_fan\_status = "Working";

**}**

### else{

**Gas\_status = "No Gas Leakage is Detected"; exhaust\_fan\_status = "Not Working";**

### }

**//json format for IBM Watson**

### String payload = "{"; payload+="\"gas\":"; payload+=gas; payload+=",";

**payload+="\"temperature\":"; payload+=(int)temperature;**

### payload+=","; payload+="\"flame\":"; payload+=flamereading; payload+=",";

**payload+="\"fire\_status\":\""+flame\_status+"\",";**

### payload+="\"sprinkler\_status\":\""+sprinkler\_status+"\ ",";

**payload+="\"Gas\_status\":\""+Gas\_status+"\",";**

### payload+="\"exhaust\_fan\_status\":\""+exhaust\_fan\_statu s+"\"}";

**if(client.publish(publishTopic, (char\*) payload.c\_str()))**

### {

**Serial.println("Publish OK");**

### }

**else{**

### Serial.println("Publish failed");

**}**

### delay(1000);

**if (!client.loop())**

### {

**mqttConnect();**

### }

**}**

### void wifiConnect()

**{**

### Serial.print("Connecting to "); Serial.print("Wifi"); WiFi.begin("Wokwi-GUEST", "", 6); while (WiFi.status() != WL\_CONNECTED)

**{**

### delay(500); Serial.print(".");

**}**

### Serial.print("WiFi connected, IP address: "); Serial.println(WiFi.localIP());

**}**

### void mqttConnect()

**{**

### if (!client.connected())

**{**

### Serial.print("Reconnecting MQTT client to ");

**Serial.println(server);**

### while (!client.connect(clientId, authMethod, token))

**{**

### Serial.print("."); delay(500);

**}**

### Serial.println();

**}**

### }