Industry-specific intelligent fire management system

SNS College of Technology, Coimbatore

Project Report

Rajendraswamy I

Ravi Kiran S

Shandeepram R

Sharanya S

Team ID: PNT2022TMID17702

Mentor : SATHISH KUMAR.R

Title **Page No**

1. **INTRODUCTION** **3**
   - 1.1 Project Overview **3**
   - 1.2 Purpose........ **3**
2. **LITERATURE SURVEY**........ **3**
   - 2.1 Existing problem........ **3**
   - 2.2 References........ **3**
   - 2.3 Problem Statement Definition........ **3**
3. **IDEATION & PROPOSED SOLUTION** **4**
   - 3.1 Empathy Map Canvas **4**
   - 3.2 Ideation & Brainstorming **5**
   - 3.3 Proposed Solution **7**
   - 3.4 Problem Solution fit........ **8**
4. **REQUIREMENT ANALYSIS** **9**
   - 4.1 Functional requirement **9**
   - 4.2 Non-Functional requirements **9**
5. **PROJECT DESIGN** **10**
   - 5.1 Data Flow Diagrams **10**
   - 5.2 Solution & Technical Architecture **11**
   - 5.3 User Stories........ **12**
6. **PROJECT PLANNING & SCHEDULING** **12**
   - 6.1 Sprint Planning & Estimation **12**
   - 6.2 Sprint Delivery Schedule **13**
   - 6.3 Reports from JIRA........ **13**
7. **CODING & SOLUTIONING**........ **15**
   - 7.1 Feature 1 .... **15**
   - 7.2 Feature 2 .... **19**
   - 7.3 Feature 3 .... **20**
8. **TESTING** **22**
   - 8.1 Test Cases........ **22**
   - 8.2 User Acceptance Testing **22**
9. **RESULTS** **23**
   - 9.1 Performance Metrics........ **23**
10. **ADVANTAGES & DISADVANTAGES**........ **25**
11. **CONCLUSION**........ **26**
12. **FUTURE SCOPE** **26**
13. **APPENDIX** **27**
    - Source Code 28
    - Github link & Demo video link 32

1. Introduction

1.1 Project Overview

The smart fire management system includes a gas, flame, and temperature sensor to detect any environmental changes. 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 the Fire station.

1.2 Purpose

- To give a detect the status of the room with IoT devices
- To turn on sprinkler and exhaust fan when there is accident
- To detect the flow of water
- To send and store the temperature status in a cloud storage
- To give a easy management system on dashboard
- To give a overview of what's happening to the user
- To send a sms to the authorities when there is a fire accident

2. Literature survey

2.1 Existing Problem

The situation is not ideal because the fire management system in houses and industries are not very reliable, efficient, cost-effective and does not have any advanced processing and does not have any features like an automatic alert system for admin and authorities and in many buildings . They are using older fire safety systems that doesn't can even activate the sprinkler system and all of they don't communicate with each other properly to prevent false alarm also monitor the entire system using applications.

2.2 Reference

https://pdfs.semanticscholar.org/f3e7/a7c0cf2d448be592421045033506e845e6c2.pdf

https://www.mdpi.com/2224-2708/7/1/11

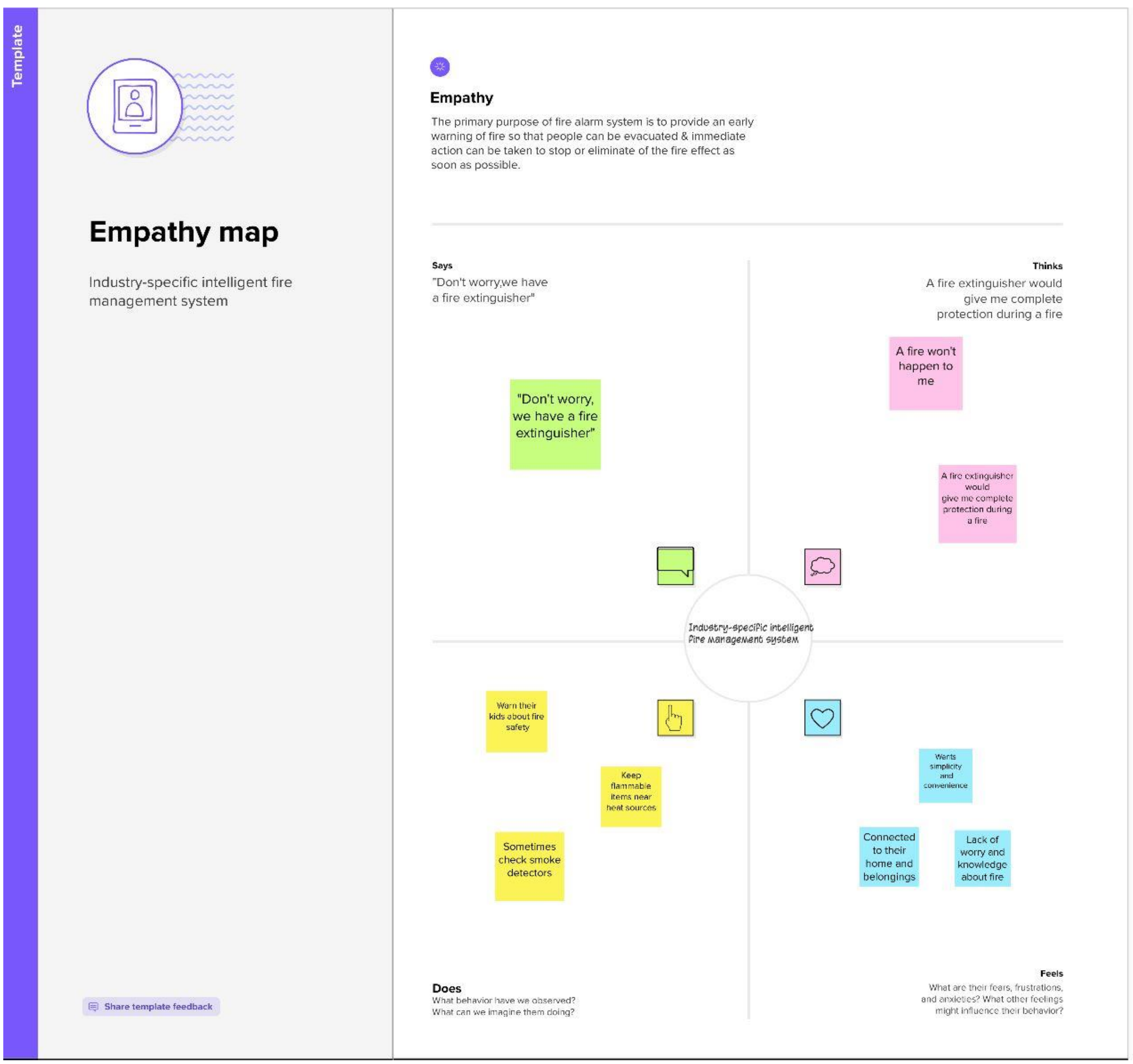
2.3 Problem Statement Definition

The fire management system in houses and industries are not very reliable ,efficient, cost effective and does not have any advance processing and does not have any features like automatic alert system for admin and authorities and in many buildings there are using older fire safety system that doesn't can even activate the sprinkler system and all of they don't communicate with each other properly to prevent false alarm also monitor the entire system using a applications .

3. Ideation and Proposed solution

3.1 Empathy map canvas

- An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes
- It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it
- The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.



3.2 Ideation and Brainstorming

step 1: Team Gathering, Collaboration and Select the Problem Statement

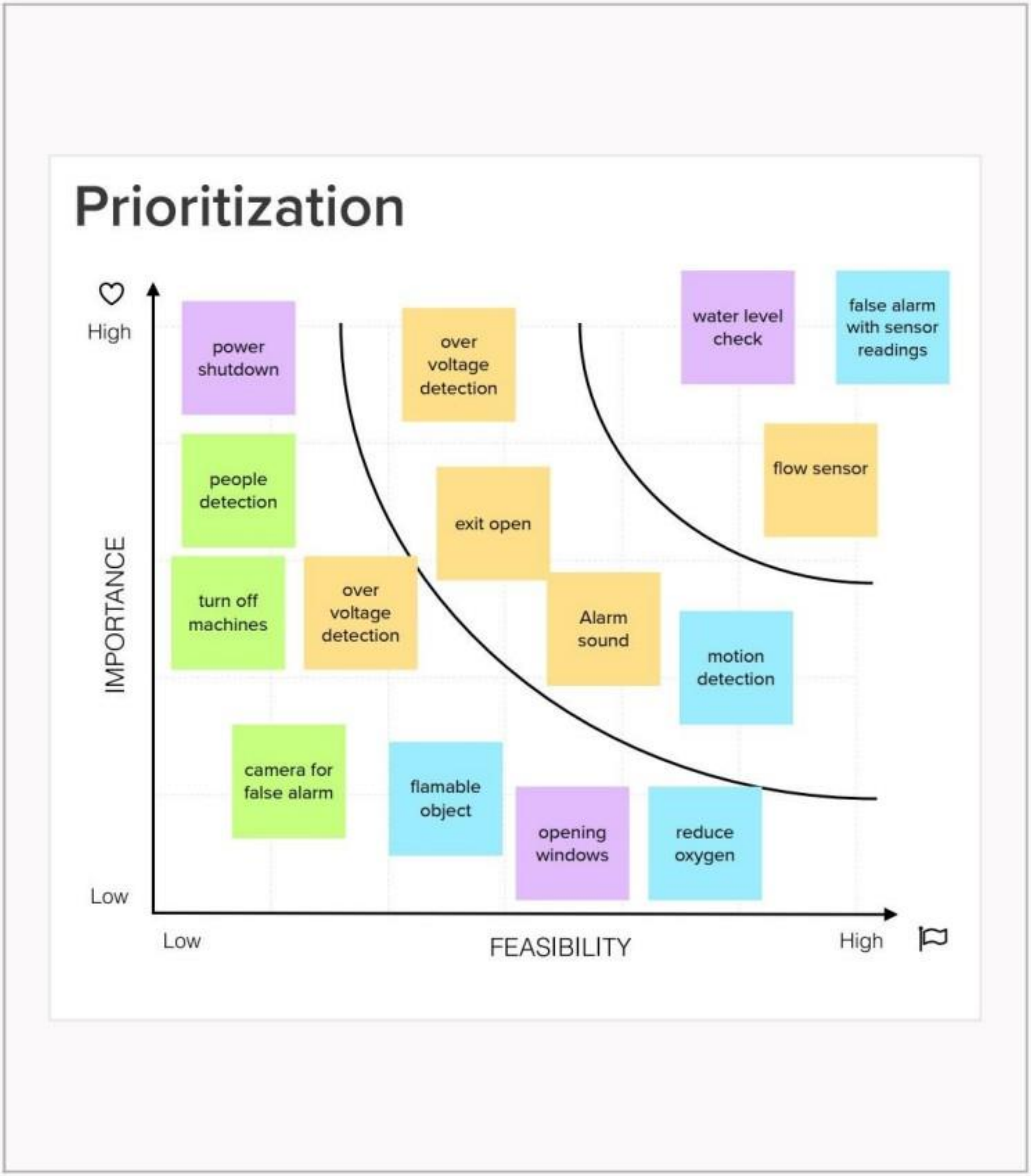
Team was gathered in mural app for collaboration

The team members are

- Rajendraswamy
- Ravi Kiran
- Shandeepram
- Sharanya

step 2:Brainstorm, Idea Listing and Grouping

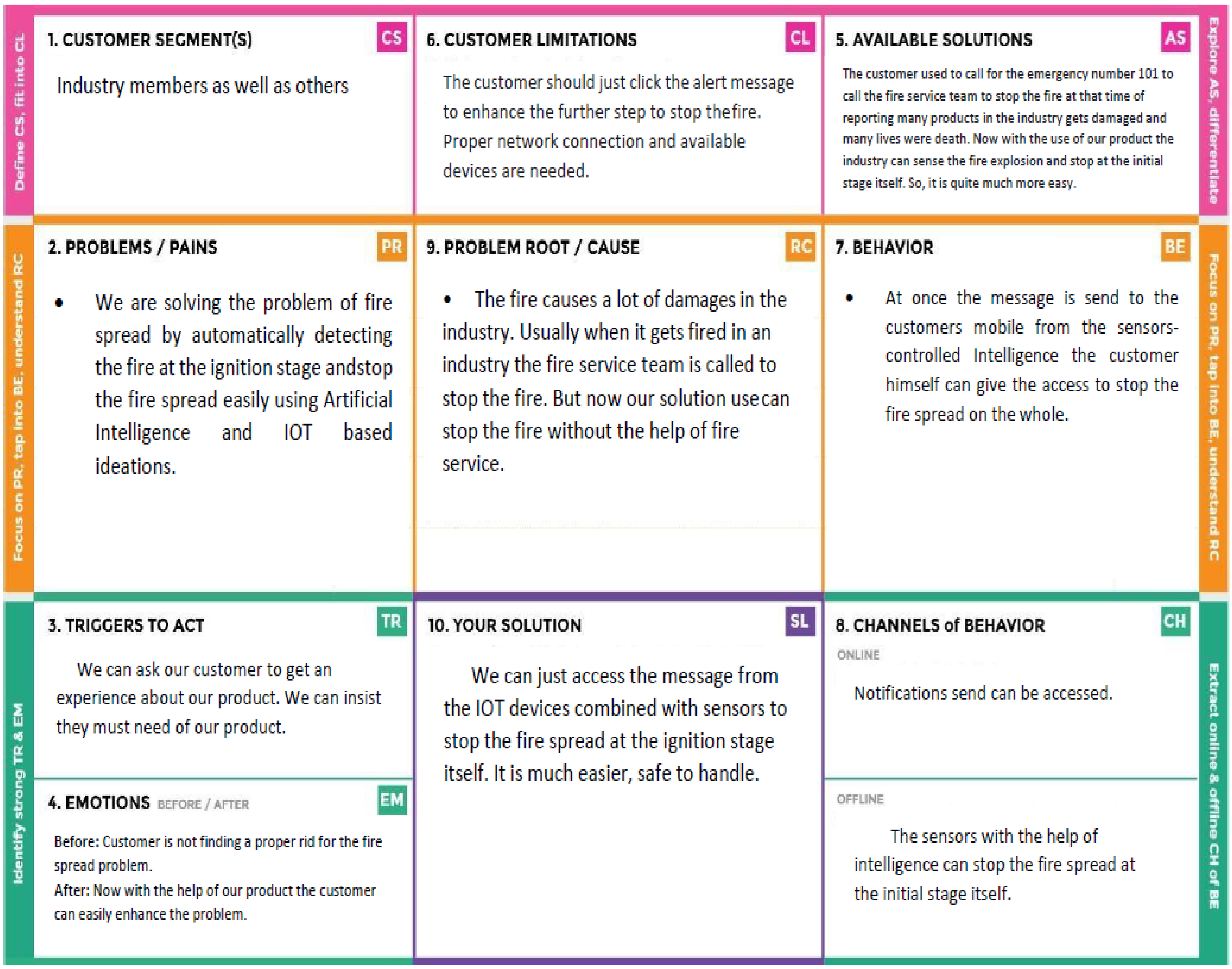
step3:Idea Prioritization



3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To create a smart industry-specific fire management system using IoT. It should have all the basic features for handling fire and report the incident to the fire department
2.	Idea / Solution description	Our solution is to provide a reliable smart fire management system that consists of exhaust fans, and sprinklers. We also ensure the proper working of sprinklers with flow sensors and check the water level for easy maintenance. It also sends periodic data to the safety sector in the company, in case of a negative situation it sends an alert to the fire department. The toxic gases and excess hydrogen and oxygen from water vapour are redirected towards outdoors by using an exhaust fan to avoid further combustion or spreading of flames by those gases.
3.	Novelty / Uniqueness	As a sprinkler gives an instant and efficient way to put down fire, we need to check the water source and the connection to it with the sprinkler this increase additional work and maintenance. This is solved by our smart system
4.	Social Impact / Customer Satisfaction	This gives a simple and powerful system making the focus and time more towards safety and not maintenance. This cuts the cost spend on maintenance, as it can be invested in other sectors.
5.	Business Model (Revenue Model)	It will cost an installation fee then the cloud and maintenance of the devices are handled in the subscription model. An additional scaling fee is also charged
6.	Scalability of the Solution	In medium or large-scale industries it is scalable. They can add any number of devices which are handled coherently in the cloud.

3.4 Proposed solution fit



4. Requirement analysis

4.1 Functional Requirements

- A functional requirement defines a function of a system or its component, where a function is
- described as a specification of behaviour between inputs and outputs.
- It specifies “what should the software system do?”
- Defined at a component level
- Usually easy to define
- Helps you verify the functionality of the software

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Device configuration	New IoT device is created in the cloud The device is configured with the new cloud device
FR-2	Admin dashboard/admin panel	Data from sensors shown in pictorial form Controls are given in the button format
FR-3	Internet connectivity	Make sure fully-fledged internet connectivity is required for smooth communication between device and cloud
FR-4	SMS API	A external SMS API is required

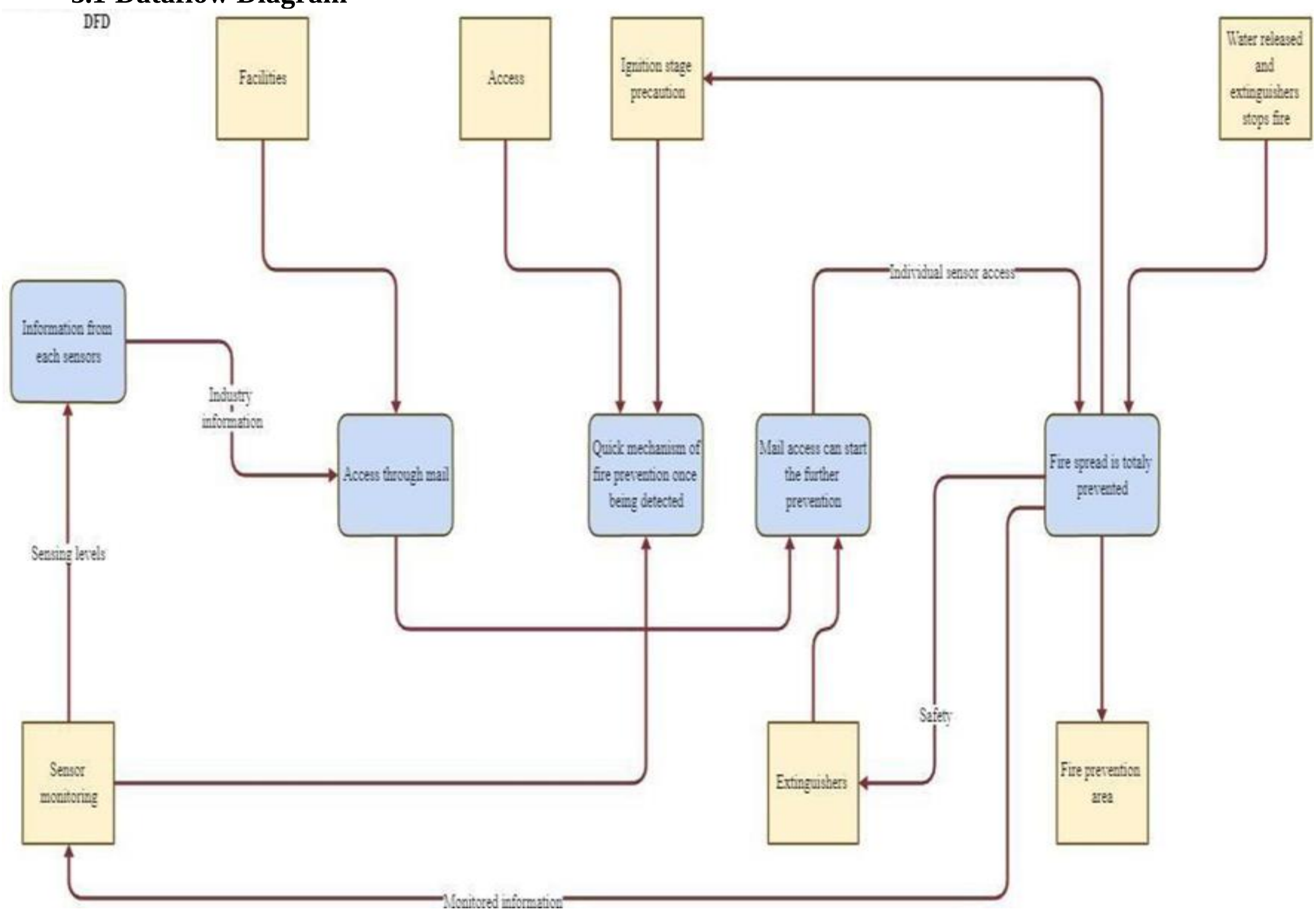
4.2 Non Functional Requirements

- A non-functional requirement defines the quality attribute of a software system
- It places constraint on “How should the software system fulfil the functional requirements?”
- It is not mandatory
- Applied to system as a whole
- Usually more difficult to define
- Helps you verify the performance of the software

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The dashboard can be used via a web browser It gives an abstract view in an easy-to-use form.
NFR-2	**Security**	As the data is sent through HTTPS the data is encrypted, so it is safe.
NFR-3	**Reliability**	The system is completely reliable as long as the internet and power is reliable
NFR-4	**Performance**	Only the data input and basic checking is done in smart device other heavy tasks are done in cloud.
NFR-5	**Availability**	The entire system is available for your service and for configuration .
NFR-6	**Scalability**	The smart system is scalable,we can add any number of devices as long as the IBM IoT platform supports it.

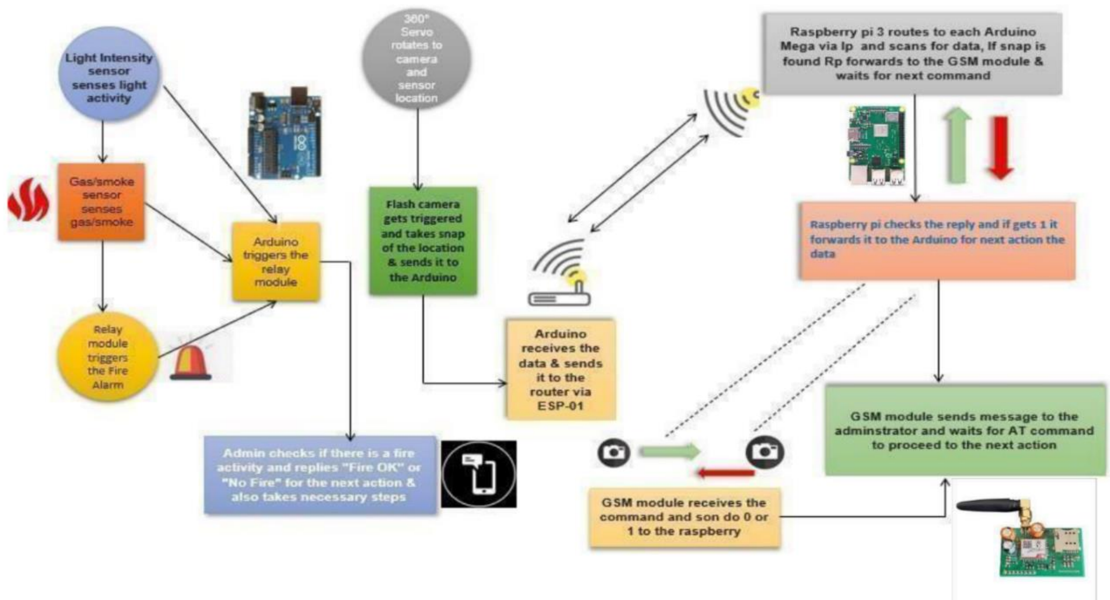
5. Project Design

5.1 Dataflow Diagram

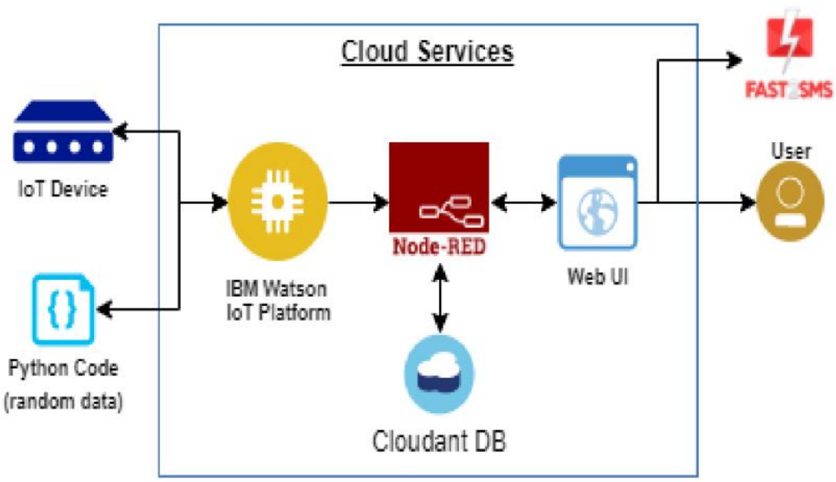


5.2 Solution and Technical architecture

Solution Architecture



Technical Architecture



5.3 User stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Monitor The Environment	USN-1	User can monitor the sensor data receiving from the microprocessor	User Can See the dashboard with sensor information	Medium	Sprint 4
	Turn on or off the sprinkler and exhaust fan.	USN-2	User can turn on / off exhaust fan and sprinkler if need in that circumstance	Can turn on / off the sprinkler and exhaust fan	Medium	Sprint 4
	Authentication	USN-2	User needed to be authenticated while turning on/off the exhaust and sprinkler system	Authenticate the user for USN-2 Fuctionality	Medium	Sprint 4
Sensing	Sensing The Environment	-USN 3	Need to Sense the environment using the sensors attached to the microprocessor	Getting Data from the sensors	High	Sprint 1
Extinguish	Actuators	USN 4	If the sensors sense the fire then the immediate next step is to turn on the exhaust fan and the sprinkler system	Extinguishing the fire	High	Sprint 1
Data	Sending data to ibm Watson Hot platform	USN 5	All the sensor Data received from the microprocessor are send to the IBM Watson Lot platform	Showing in the Watson Dashboard	Medium	Sprint 2
	Node-red	USN 6	Sending the data to further process in the cloud for storing and alert purpose		High	Sprint 3
	Data Storing	USN 7	All the sensor values are stored in an cloud database	Storing the data	Low	Sprint 3
Notification	Event notification	USN 8	FIre alertMessage will send to fire department	Notifying the authorities	High	Sprint 4

6. Project design and planning

6.1 Sprint planning and estimation

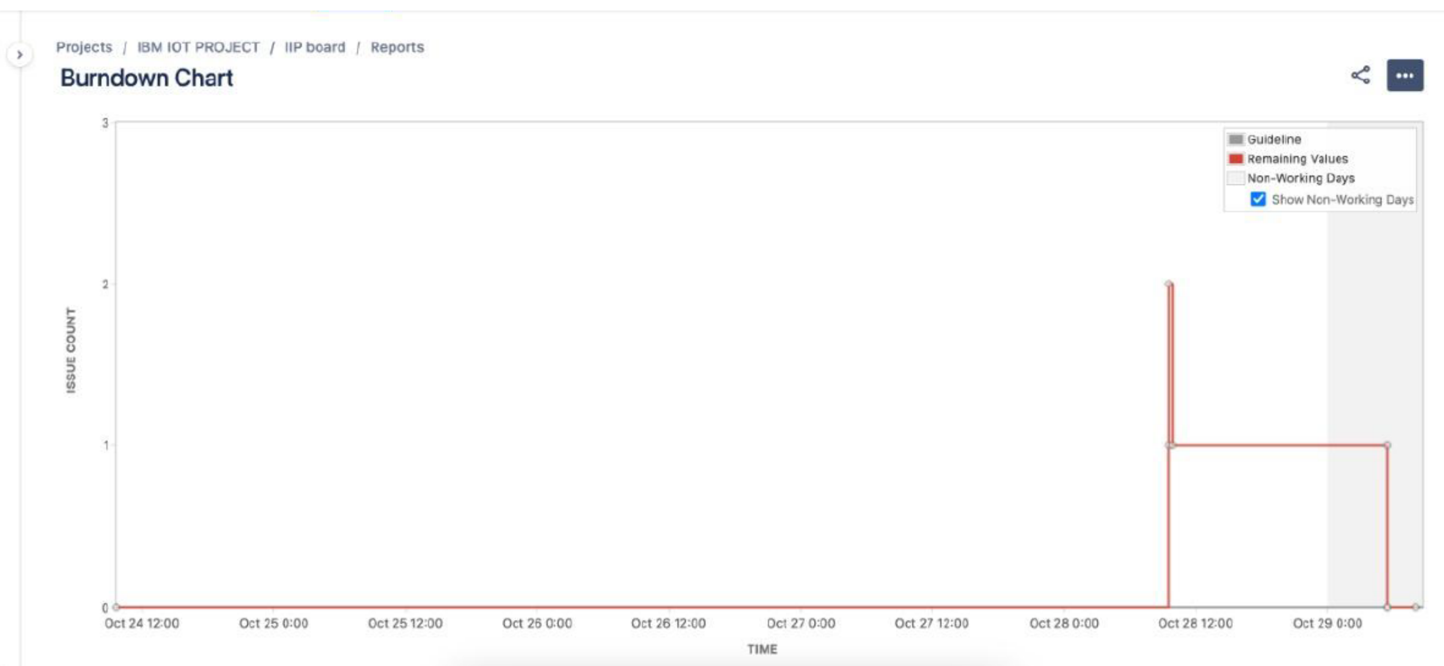
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Sensing	USN-3	Sensing the surrounding environment using the sensors	2	High	Ravi kiran
Sprint-1	Extinguish	USN-4	Turning on the exhaust fan as well as the fire sprinkler system in cause of fire	2	High	Sharanya
Sprint-2	Sending Data to the ibm Not platform	USN-5	Sending the data of the sensor form the microcontroller to the IBM Watson Dot platform	1	Medium	Rajendr aswamy
Sprint-3	Node-red	USN-6	Sending the data from the ibm Watson to the node-red for further process the data	3	High	Ravi kiran
	Storing of sensor data	USN-7	Storing the received sensor data in a cloud Database	1	Low	Shande epram
Sprint-4	Monitoring the environment	-USN 1	User can monitor the situation of the environment from a dashboard that displays sensor information about the environment	1	Medium	Sharanya

6.2 Sprint delivery schedule

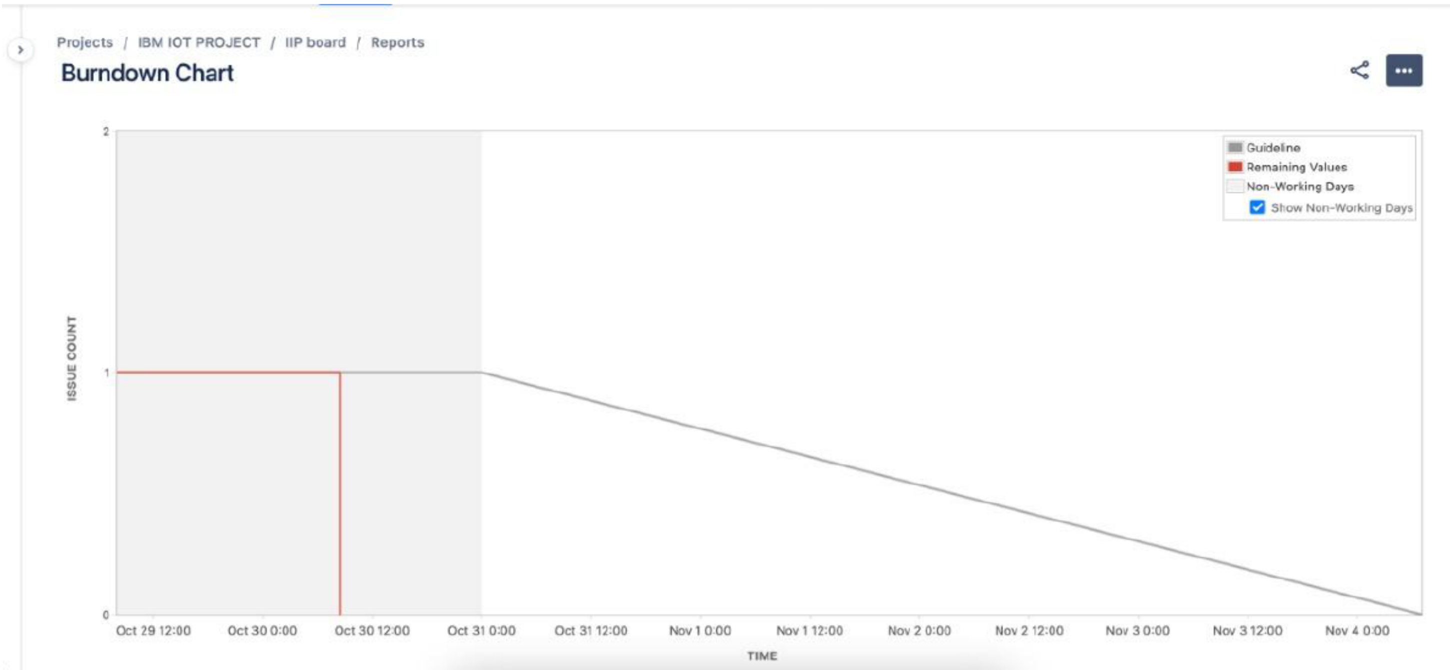
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	login	USN-1	As a user, I can register for the application by entering my email. password, and confirming my password.	2	High	Rajendraswamy
Sprint-1	Registration	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Ravi kiran
Sprint-2	Dashboard	USN-3	As a user, I can register for the application through Facebook	2	Low	Shandeepram
Sprint-1	Dashboard	USN-4	As a user, I can register for the application through Gmail	2	Medium	Sharanya

6.3 Reports

Sprint 1



Sprint 2



Sprint 3

Sprint 4

7. Coding and Solutioning

Feature 1 : Temparature reading

```
#include <WiFi.h>
#include <PubSubClient.h>
#define temp_pin 15
void callback(char* subscribetopic,byte* payload, unsigned int payloadLength);
#define ORG "0va7j8"
#define DEVICE_TYPE "esp32"
#define DEVICE_ID "1234"
#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);




 // should match the Beta Coefficient of the thermistor

void setup() {
  Serial.begin(9600);
  analogReadResolution(10);
  pinMode(32,INPUT);
  pinMode(14,OUTPUT);

  wificonnect();
  mqttconnect();

}
void loop() {
  const float BETA = 3950; // should match the Beta Coefficient of the thermistor
int analogValue = analogRead(A4);
float temp = 1 / (log(1 / (1023. / analogValue - 1)) / BETA + 1.0 / 298.15) - 273.15;
  //float temp = 1 / (log(1 / (1023. / analogValue - 1)) / BETA + 1.0 / 298.15) -
273.15;
  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println(" ℃");
  if(temp>=35){
    PublishData2(temp);
    digitalWrite(14, HIGH);
  }else{
    digitalWrite(14, LOW);
    PublishData1(temp);
}
 delay(1000);
  if(!client.loop()){
    mqttconnect();
  }

  //delay(2000);
}
void PublishData1(float tem){
  mqttconnect();
  String payload= "{\"temp\":";
  payload += tem;
```

```
  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 tem){
  mqttconnect();
  String payload= "{\"ALERT\":";
  payload += tem;
  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];
  }
  Serial.println("data:"+ data3);
  if(data3=="lighton"){
    Serial.println(data3);
    digitalWrite(14,HIGH);
  }else{
    Serial.println(data3);
    digitalWrite(14,LOW);
  }
  data3="";
}
```

Explanation

- This reads out Temperature.
- It also sets the current status.
- This also handles the permission management of whether a device would work or not.

Feature 2

```
#include <WiFi.h>
#include <PubSubClient.h>
#define temp_pin 15
void callback(char* subscribetopic,byte* payload, unsigned int payloadLength);
#define ORG "0va7j8"
#define DEVICE_TYPE "esp32"
#define DEVICE_ID "1234"
#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);


     }
```

Explanation

- It sends the data to IBM IoT Watson platform

Feature 3

```
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)==137153){
        const char *buf;
        int len;
```

```
        if (mjson_find(s, strlen(s), "$.command", &buf, &len))
        {
          String command(buf,len);
          if(command=="\"cantfan\""){
                        canfanoperate = !canfanoperate;
          }
          else if(command=="\"cantsprink\""){
            cansprinkoperate = !cansprinkoperate;
          }else if(command=="\"sentalert\""){
            resetcooldown();
          } }  } }
  data3="";
;
}
```

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

8. TESTING

8.1 Testcases

Date	12-Nov-22
Team ID	PNT2022TMID17702
Project Name	Industry-specific intelligent fire
Maximum Marks	4 marks

Test case ID	Feature Type	Component	Test Scenario	Pre-Requisite	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Commnets	TC for Automation(Y/N)	BUG ID	Executed By
Sensor_OO1	Functional	Microcontroller	Sensor data is properly taken	The connections to the	1.Open the simulator in wokwi.	Random values	Get the values and print it in the	Working as	Pass		N		Ravikiran
Sensor_OO2	Functional	Microcontroller	Sensor data is parsed as json	The microcontroller should	1.Open the simulator in wokwi.	Random values	Get the values and print it in the	Working as	Pass		N		Ravikiran
Work_001	Functional	Microcontroller	To check for fake alarm	The sensor values are taken	1.Simulate the device(do a	Random values	Accident status is properly	Working as	Pass		N		Sharanya
Work_002	Functional	Microcontroller	The data should be sent to IBM	The device setup is	1.Start the simulation in wokwi.	Random values	The values are shown in recent	Working as	Pass		N		Sharanya
Work_OO3	Functional	Node-red	The data should be sent to	The necessary packages	1.Login to node red editor	values got from the iot	The debug area should show the	Working as	Pass		N		Rajendraswamy
Work_OO4	Functional	Node-red	Verify that the json data is	A configured node-red with	1.Login to node red editor	values got from the iot	the debug menu shows the	Working as	Pass		N		Rajendraswamy
Database_001	Storage	Kubernetes	The received data is stored in database in a key value pair	The node red is connected with cloudant node	1.login to Kubernetes dashboard. 2.create new database. 3. connect the database with node red and then give the database name in required field	values got from the iot device	After sending the data the data is stored in cloudant	Working as expected	Pass		N		Shandeepram
SMS_001	API	sms API	The sms is sent when there is fire alert	The node red should be configured to send a post request	1.Simualte the fire in the simulator(if real hardware is used real fire is used). 2.or click the sent alert button in	"Fire alert at xyz industries Hurry" And the trigger inputs	sms receiving to the given phoner	Working as expected	Pass		N		Shandeepram
Work_005	Functional	UI	Even at times of emergency sometimes manual control is required	the dashboard interaction elements is connected to the node-red	1. in the dashboard enter the correct pin 2.click the action to be done	The action by user	manual command system works o	Working as expected	Pass		N		Sharanya
Auth_001	Functional	UI	Verify that the correct pin is entered	text filed is given in dashboard to enter pin	1.The correct pin is entered 2.then necessary action is required	1234	command is sent successfull	working as expected	Pass		N		Ravikiran
Auth_002	Functional	UI	Verify that it handles when wrong pin is entered	text filed is given in dashboard to enter pin	1.The correct pin is entered 2.then necessary action is required	141324 63363 1 001 fds	Show a message that the entered pin is wrong	Working as expected	Pass		N		Rajendraswamy
SMS_002	Functional	Microcontroller	Verify that the message is not sent continuously when there is fire it sends a message then waits for 10 minutes even after that if the fire exists it sends again	the sms funtionality should be implemented	1.Simulate a fire accident scenario 2.or click the send alert button on the dashboard 3.wait for the message to be sent	the event is simulated or triggered	The service should not spam continuous messages to authorities as fire won't be down within fraction of seconds	Working as expected	Pass		N		Sharanya

8.2UAT

Defect analysis

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	9	0	2	1	12
External	0	0	1	0	1
Fixed	19	24	25	14	82
Not Reproduced	0	0	2	0	2
Skipped	0	0	0	0	0
Won't Fix	0	0	0	0	0
Totals	28	24	30	15	97

Test case analysis

Section	Total Cases	Not Tested	Fail	Pass
Client Application	4	0	0	4
Security	2	0	0	2
Exception Reporting	11	0	0	11
Final Report Output	5	0	0	5

9. Results

9.1 performance metrics

CPU usage

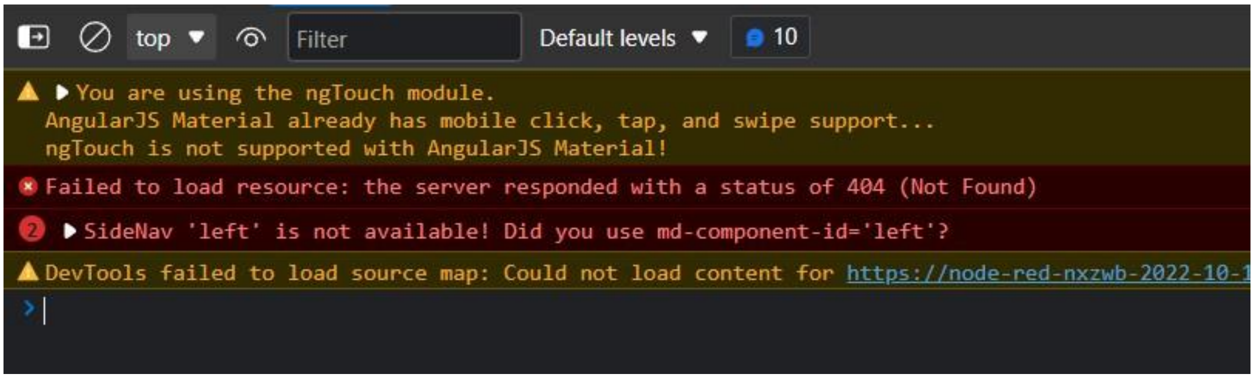
The micro version of c++ is make the best use of the CPU. For every loop the program runs in O(1) time, neglecting the network and communication. The program sleeps for every 1 second for better communication with MQTT. As the program takes O(1) time and the compiler optimizes the program during compilation 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

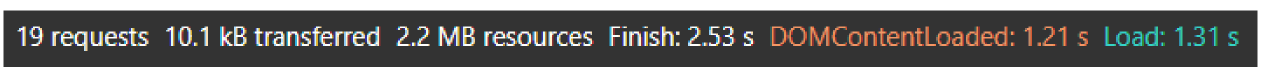
Error rates :

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

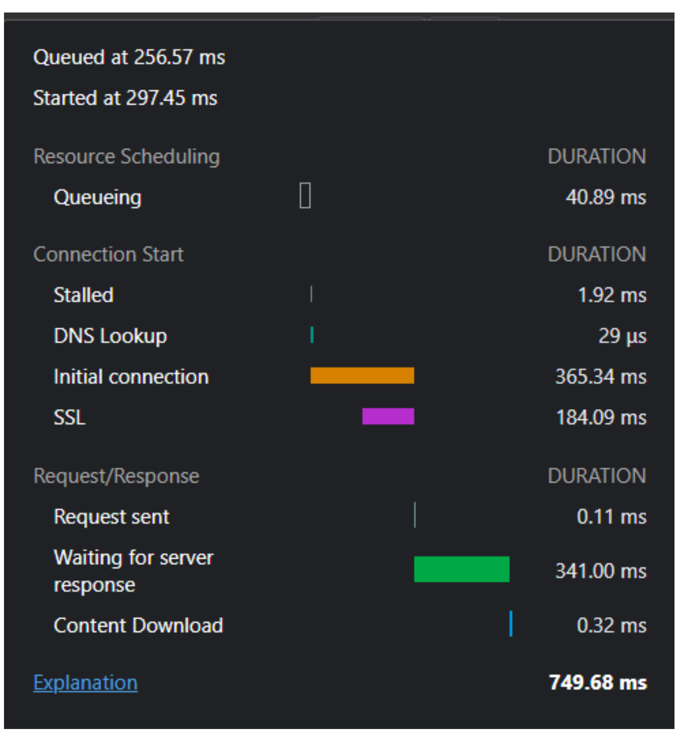


Latency and Response Time :

The DOM handling of the received data is optimal and latency is low .After the DOM is loaded the entire site is loaded to the browser



The server also responses quickly . The average time of response is respectable

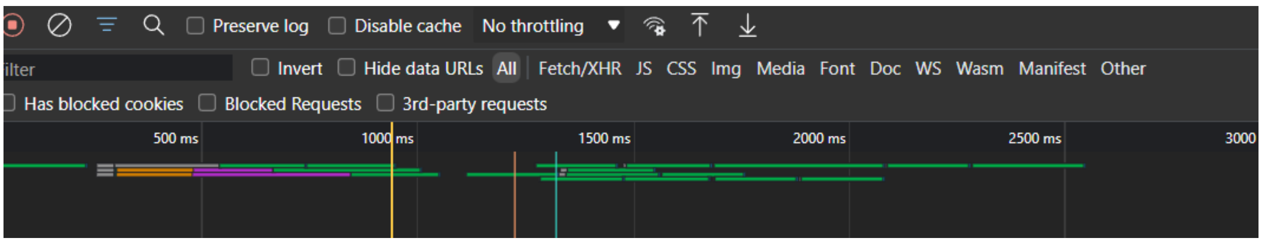


For the data sent from the IoT device (considering the sleep of one second from the IoT), the response is much quicker .We can easily see the delay caused by the sleep function

The average time is well over optimal value

$$\text{Averagetime} = (5ms + 2600ms)/2$$

$$= 1302.5$$



Garbage collection:

In the server-side garbage collection is done by the Node framework.In the IoT device , c++ does not have any garbage collection features . 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.

10. Advantages and Disadvantages

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

11. 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 .

12. 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.

13. APPENDIX

Esp32 - Microcontroller :

ESP32 is **a highly-integrated solution for Wi-Fi-and-Bluetooth IoT applications, with around 20 external components**. ESP32 integrates an antenna switch, RF balun, power amplifier, low-noise receive amplifier, filters, and power management modules.

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).

Flow Sensors

A flow sensor (more commonly referred to as a “flow meter”) is an electronic device that measures or regulates the flow rate of liquids and gasses within pipes and tubes.

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:

```
#include <WiFi.h>
#include <PubSubClient.h>
#define temp_pin 15
void callback(char* subscribetopic,byte* payload, unsigned int payloadLength);
#define ORG "0va7j8"
#define DEVICE_TYPE "esp32"
#define DEVICE_ID "1234"
#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);



 // should match the Beta Coefficient of the thermistor

void setup() {
  Serial.begin(9600);
  analogReadResolution(10);
  pinMode(32,INPUT);
  pinMode(14,OUTPUT);

  wificonnect();
  mqttconnect();

}
void loop() {
  const float BETA = 3950; // should match the Beta Coefficient of the thermistor
```

```
int analogValue = analogRead(A4);
float temp = 1 / (log(1 / (1023. / analogValue - 1)) / BETA + 1.0 / 298.15) -
273.15;
  //float temp = 1 / (log(1 / (1023. / analogValue - 1)) / BETA + 1.0 / 298.15) -
273.15;
  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println(" ℃");
  if(temp>=35){
    PublishData2(temp);
    digitalWrite(14, HIGH);
  }else{
    digitalWrite(14, LOW);
    PublishData1(temp);
}
 delay(1000);
  if(!client.loop()){
    mqttconnect();
  }

  //delay(2000);
}
void PublishData1(float tem){
  mqttconnect();
  String payload= "{\"temp\":";
  payload += tem;
  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 tem){
  mqttconnect();
  String payload= "{\"ALERT\":";
  payload += tem;
  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];
  }
  Serial.println("data:"+ data3);
  if(data3=="lighton"){
    Serial.println(data3);
    digitalWrite(14,HIGH);
  }else{
    Serial.println(data3);
    digitalWrite(14,LOW);
  }
  data3="";
}

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

void PublishData(float temp, int gas ,int flame ,int flow,bool
isfanon,bool issprinkon) {

  mqttconnect();//function call for connecting to ibm

  /*

     creating the String in in form JSon to update the data to ibm
cloud



import wiotp.sdk.device
import time
from collections.abc import MutableSequence
import random

myConfig = {
    "identity": {
        "orgId": "0va7j8",
        "typeId": "esp32",
        "deviceId":"1234"
    },
    "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:
    temperature = random.randint(-20,125)
    gas = random.randint(0,1000)
    flamereading = random.randint(200,1024)
    myData={'Temperature': temperature, 'Gas':gas, 'Flamereading':flamereading}
    print("Published data Successfully: %s", myData)
    client.commandCallback = myCommandCallback
    time.sleep(2)
    client.disconnect()
```

Github Link : **https://github.com/IBM-EPBL/IBM-Project-45186-1660728724**

Demo Video : **https://youtu.be/FD4WANqL5Gc**