

# Industry-specific intelligent fire management system

## Using Wokwi IOT Platform To MIT App

Team ID	PNT2022TMID27962
Project Name	Industry Specific intelligent fire management system

### Wokwi Code:

```
#include <WiFi.h> //library for wifi
#include <PubSubClient.h> //library for MQTT
#include "DHT.h" // Library for dht11
#define DHTPIN 15
#define dhtpin 13 // what pin we're connected to
#define DHTTYPE DHT22 // define type of sensor DHT 11
#include <ESP32Servo.h>

const int servoPin = 18;
Servo servo;

DHT dht1 (DHTPIN, DHTTYPE); // creating the instance by passing pin and typr of
dht connected
DHT dht2 (dhtpin, DHTTYPE);
void callback(char* subscribetopic, byte* payload, unsigned int
payloadLength);

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

#define ORG "2piqlm" //IBM ORGANITION ID
#define DEVICE_TYPE "Code" //Device type mentioned in ibm watson IOT Platform
#define DEVICE_ID "123456" //Device ID mentioned in ibm watson IOT Platform
#define TOKEN "12345678" //Token
String data3;

float g , t;
float fl,T;
int f;
int pos;

//----- 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
float dist,dur;
String data;
//-----
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);
    dht1.begin();
    dht2.begin();
    delay(10);
    Serial.println();
    servo.attach(servoPin, 500, 2400);
    wificonnect();
    mqttconnect();
}

void loop()// Recursive Function
{

    g = dht1.readHumidity();
    t = dht1.readTemperature();
    fl = dht2.readHumidity();

    Serial.print("temp:");
    Serial.println(t);
    Serial.print("gas:");
    Serial.println(g);
    Serial.print("flame:");
    Serial.println(fl);
    if(g>=50)
    {
        Serial.print("Gas Detected , Fan ON");
        f=1;
        for (pos = 0; pos <= 180; pos += 1) {
            servo.write(pos);
            delay(15);
        }
    }
}

```

```

    for (pos = 180; pos >= 0; pos -= 1) {
        servo.write(pos);
        delay(15);
    }

}

if(fl>=90)
{
    Serial.print("Flame Detected , Sprinkler ON");
    f=1;
    for (pos = 0; pos <= 180; pos += 1) {
        servo.write(pos);
        delay(15);
    }
    for (pos = 180; pos >= 0; pos -= 1) {
        servo.write(pos);
        delay(15);
    }
}

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

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

void PublishData(float temp, float humid, int flame) {
    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 += temp;
    payload += "," " \"gas\":";
    payload += humid;
    payload += "," " \"flame\":";
    payload += flame;
    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 PublishAlert() {
    mqttconnect();//function call for connecting to ibm
    /*
        creating the String in in form JSon to update the data to ibm cloud
    */
    String payload = "{\"alert\":";
    payload += 10000;
    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)
{
}

```

## Wokwi Circuit:

WOKWI

SAVE

SHARE

sketch.ino copy

Docs

sketch.ino

diagram.json

libraries.txt

Library Manager

Simulation

▶

+

⋮

```

1  #include <WiFi.h> //library for wifi
2  #include <PubSubClient.h> //library for MQTT
3  #include "DHT.h" // Library for dht11
4  #define DHTPIN 15
5  #define dhtpin 13 // what pin we're connected to
6  #define DHTTYPE DHT22 // define type of sensor DHT 11
7  #include <ESP32Servo.h>
8
9  const int servoPin = 18;
10 Servo servo;
11
12 DHT dht1 (DHTPIN, DHTTYPE); // creating the instance by passing pin and type of
13 DHT dht2 (dhtpin, DHTTYPE);
14 void callback(char* subscribetopic, byte* payload, unsigned int payloadLength)
15
16 //-----credentials of IBM Accounts-----
17
18 #define ORG "2piqlm" //IBM ORGANITION ID
19 #define DEVICE_TYPE "Code" //Device type mentioned in ibm watson IOT Platform
20 #define DEVICE_ID "123456" //Device ID mentioned in ibm watson IOT Platform
21 #define TOKEN "12345678" //Token
22 String data3;
23
24 float g, t;
25 float fl, T;

```

## WOKWI OUTPUT:

WOKWI interface showing a sketch and its simulation output.

**Sketch (sketch.ino):**

```
73 f=1;
74 for (pos = 0; pos <= 180; pos += 1) {
75   servo.write(pos);
76   delay(15);
77 }
78 for (pos = 180; pos >= 0; pos -= 1) {
79   servo.write(pos);
80   delay(15);
81 }
82 }
83 }
84 if(f1>90)
85 {
86   Serial.print("Flame Detected , Sprinkler ON");
87   f=1;
88   for (pos = 0; pos <= 180; pos += 1) {
89     servo.write(pos);
90     delay(15);
91   }
92   for (pos = 180; pos >= 0; pos -= 1) {
93     servo.write(pos);
94     delay(15);
95   }
96 }
97 }
```

**Simulation Output:**

```
temp:58.00
gas:43.00
flame:64.00
Sending payload: {"temp":58.00,"gas":43.00,"flame":64}
Publish ok
temp:58.00
gas:43.00
flame:64.00
Sending payload: {"temp":58.00,"gas":43.00,"flame":64}
Publish ok
temp:58.00
gas:43.00
flame:99.50
Flame Detected , Sprinkler ON
Sending payload: {"temp":58.00,"gas":43.00,"flame":99}
Publish ok
temp:58.00
gas:43.00
flame:99.50
```

WOKWI interface showing a sketch and its simulation output.

**Sketch (sketch.ino):**

```
73 f=1;
74 for (pos = 0; pos <= 180; pos += 1) {
75   servo.write(pos);
76   delay(15);
77 }
78 for (pos = 180; pos >= 0; pos -= 1) {
79   servo.write(pos);
80   delay(15);
81 }
82 }
83 }
84 if(f1>90)
85 {
86   Serial.print("Flame Detected , Sprinkler ON");
87   f=1;
88   for (pos = 0; pos <= 180; pos += 1) {
89     servo.write(pos);
90     delay(15);
91   }
92   for (pos = 180; pos >= 0; pos -= 1) {
93     servo.write(pos);
94     delay(15);
95   }
96 }
97 }
```

**Simulation Output:**

```
gas:60.50
Reconnecting client to
2piqlm.messaging.internetofthings.ibmcloud.com
iot-2/cmd/command/fmt/String
subscribe to cmd OK
Sending payload: {"temp":58.00,"gas":60.50,"flame":64}
Publish ok
temp:58.00
gas:60.50
flame:64.00
Gas Detected , Fan ON
Sending payload: {"temp":58.00,"gas":60.50,"flame":64}
Publish ok
temp:58.00
gas:60.50
flame:64.00
Gas Detected , Fan ON
```

## WOKWI TO IBM IOT WATSON:

The screenshot shows the IBM Watson IoT Platform interface. The top navigation bar includes 'Browse', 'Action', 'Device Types', and 'Interfaces'. A sidebar on the left contains icons for various functions. The main content area displays details for device '123456', which is 'Connected'. The 'Recent Events' tab is active, showing a table of events. The table has columns for 'Event', 'Value', 'Format', and 'Last Received'. The events are JSON payloads containing temperature, gas, and flame data. A message '0 Simulations running' is visible at the bottom right of the events table.

Event	Value	Format	Last Received
Data	{"temp":58,"gas":60.5,"flame":64}	json	a few seconds ago
Data	{"temp":58,"gas":40,"flame":64}	json	a few seconds ago
Data	{"temp":58,"gas":40,"flame":64}	json	a few seconds ago
Data	{"temp":58,"gas":40,"flame":64}		
Data	{"temp":58,"gas":40,"flame":64}		

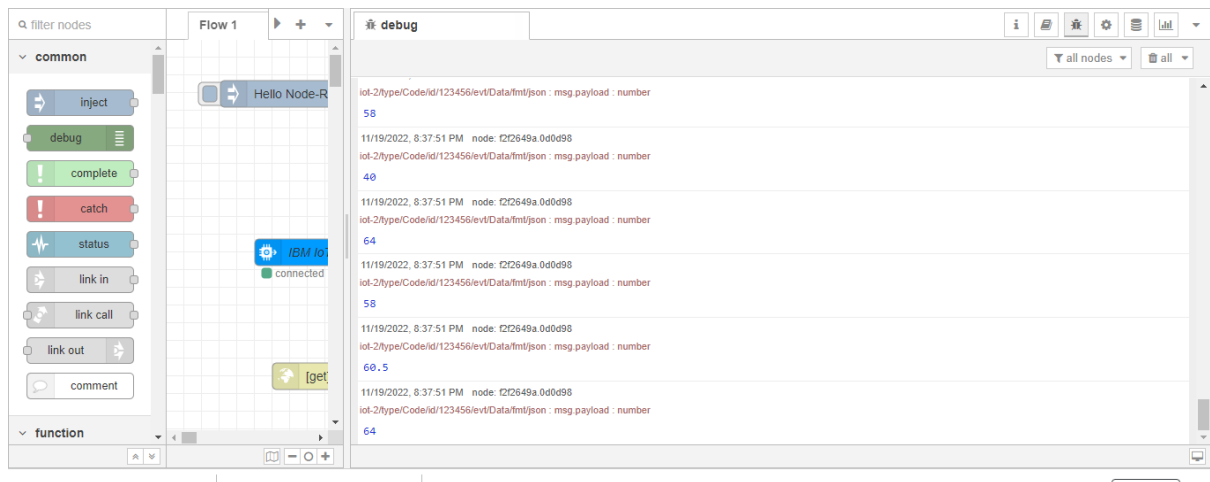
## IBM WATSON TO NODE RED:

The screenshot shows the Node-RED interface. The 'Flow 1' workspace contains a 'Hello Node-RED!' node, an 'IBM IoT' node (labeled 'connected'), and three function nodes labeled 'Temperature', 'Gas', and 'Flame'. These function nodes are connected to a 'msg.payload' node. Below this, there is a '[get] /sensor' node connected to a 'function' node, which is then connected to an 'http' node. The right sidebar shows the 'debug' console with a list of messages, including the JSON payloads from the IoT device.

```
graph LR
    Hello[Hello Node-RED!]
    IoT[IBM IoT]
    Temp[Temperature]
    Gas[Gas]
    Flame[Flame]
    Payload[msg.payload]
    Sensor[/sensor/]
    Function[function]
    HTTP[http]

    IoT --> Temp
    IoT --> Gas
    IoT --> Flame
    Temp --> Payload
    Gas --> Payload
    Flame --> Payload
    Sensor --> Function
    Function --> HTTP
```

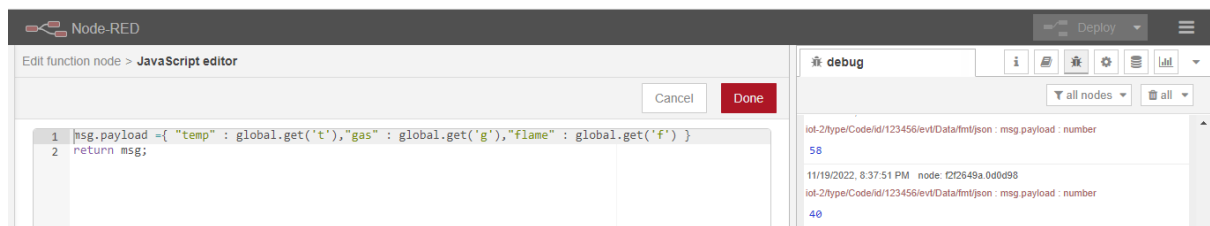
## NODE RED OUTPUT:



The screenshot shows the Node-RED web interface. On the left, the 'common' node palette is visible, containing nodes like inject, debug, complete, catch, status, link in, link call, link out, and comment. The 'function' node palette is also visible. In the center, a flow named 'Flow 1' contains a 'Hello Node-R' node and an 'IBM IoT' node. On the right, the 'debug' node is selected, and its console shows a series of log messages. The messages are as follows:

Timestamp	Node ID	Message
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	58	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	40	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	64	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	58	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	60.5	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	64	

## NODE RED FUNCTION CODE:



The screenshot shows the Node-RED web interface with the 'JavaScript editor' open. The editor contains the following code:

```
1 msg.payload = { "temp" : global.get('t'), "gas" : global.get('g'), "flame" : global.get('f') }  
2 return msg;
```

The 'debug' node is selected, and its console shows the output of the function. The messages are as follows:

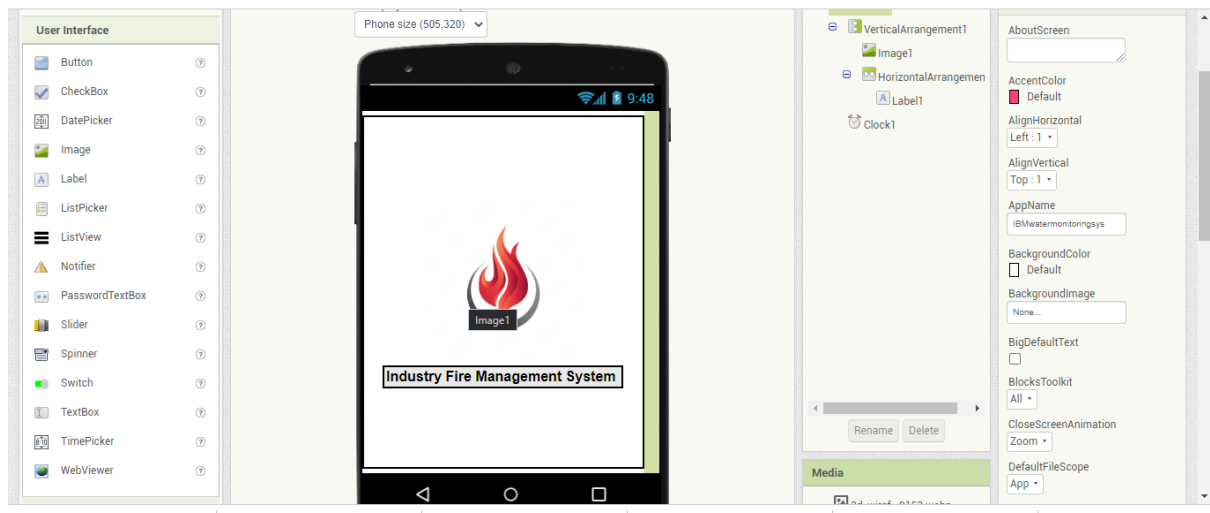
Timestamp	Node ID	Message
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	58	
11/19/2022, 8:37:51 PM	node: f2f2649a-0d0d98	iot-2/type/CodeId/123456/evt/Data/fmt/json : msg.payload : number
	40	



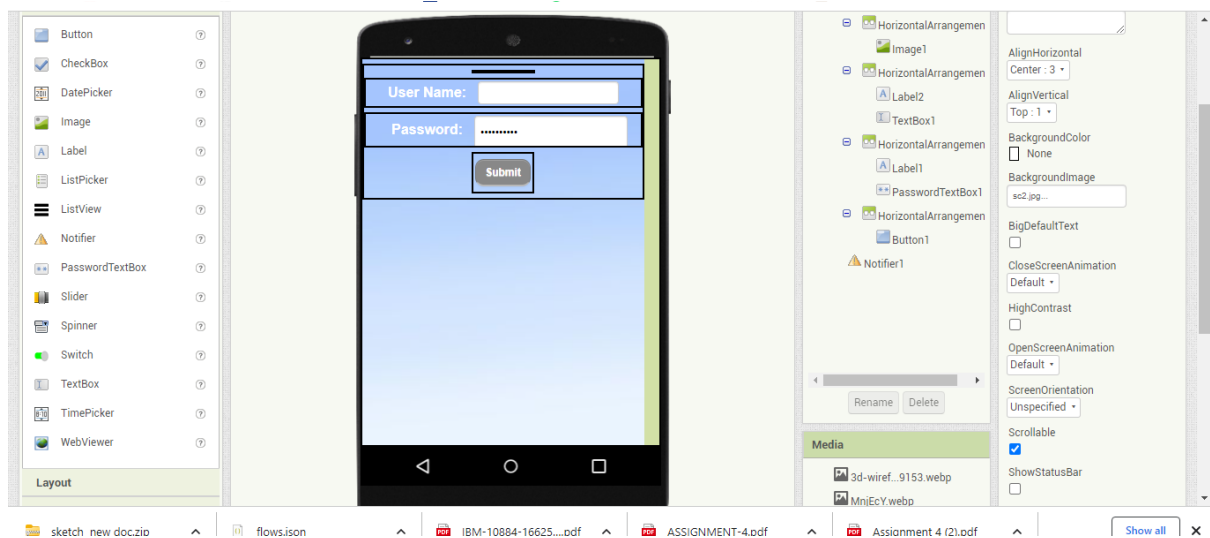
## NODE RED TO MIT APP:

## APP CREATION: FRONTEND

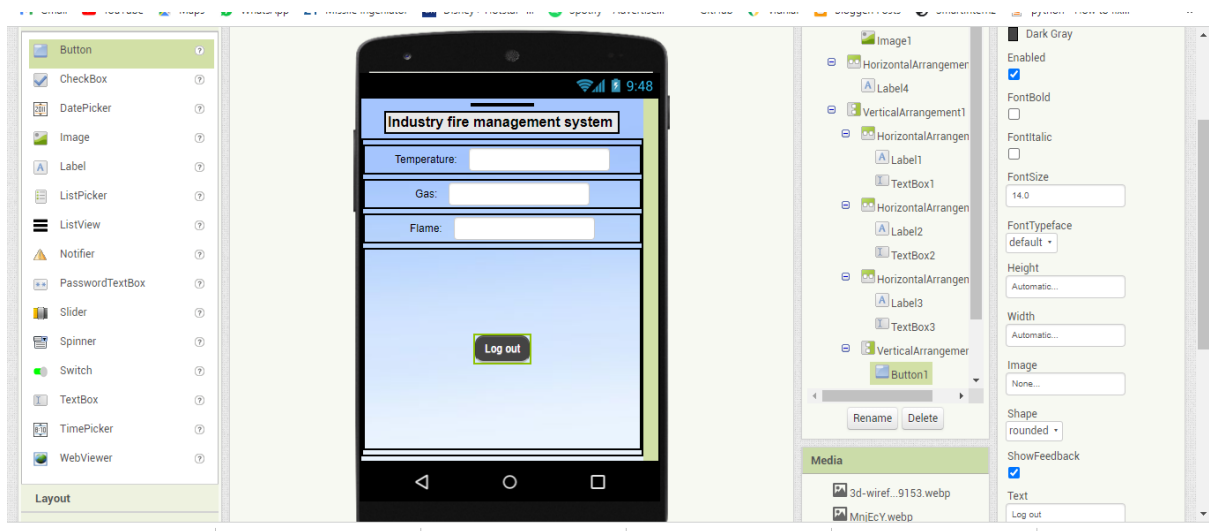
### SCREEN 1:



### SCREEN 2:

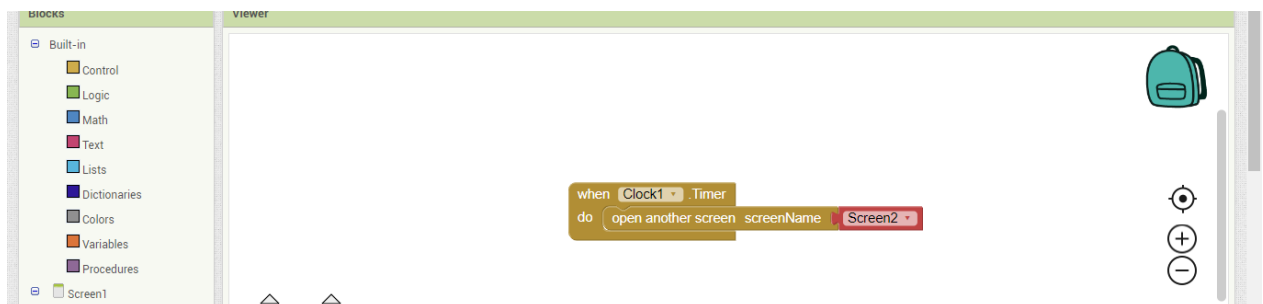


## SCREEN 3:

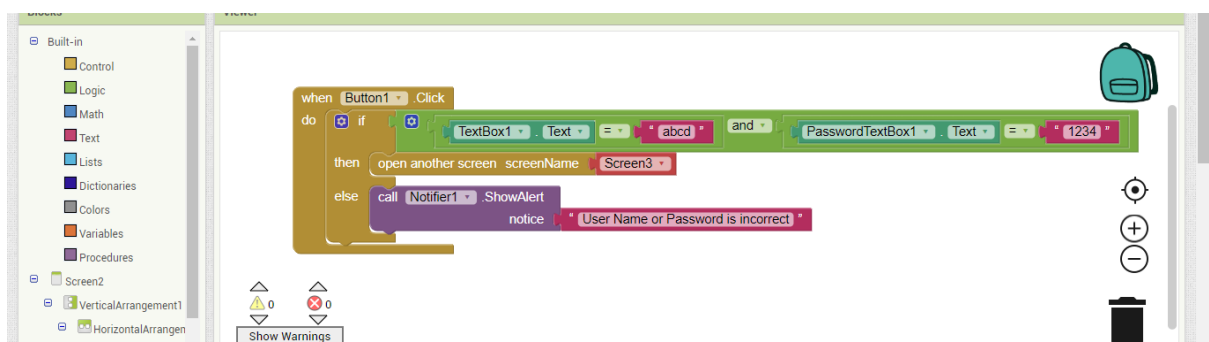


## BACKEND BLOCKS:

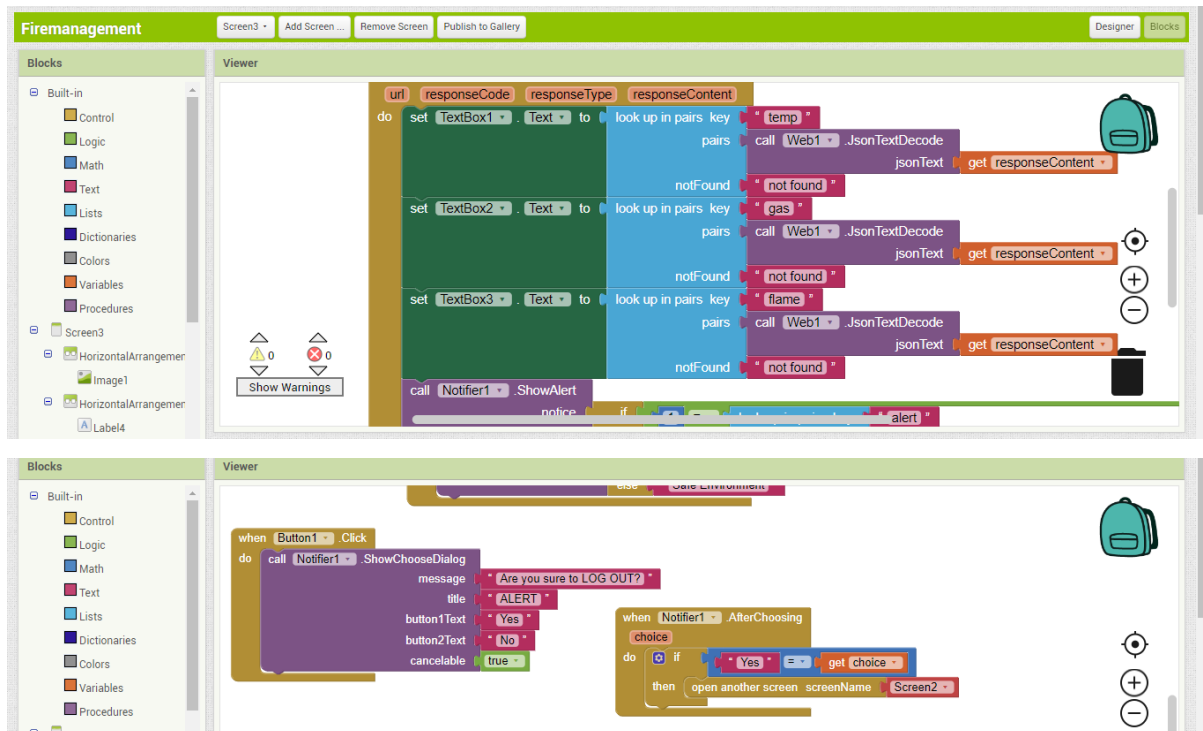
### SCREEN 1:



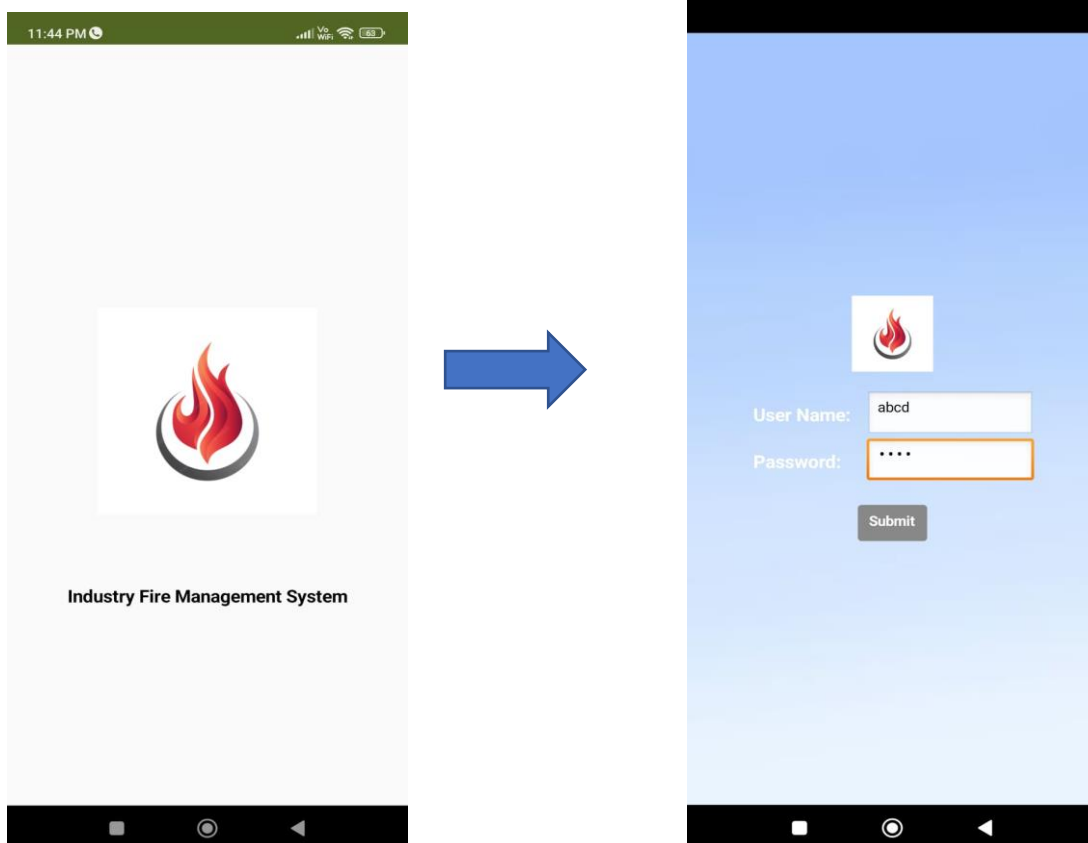
### SCREEN 2:

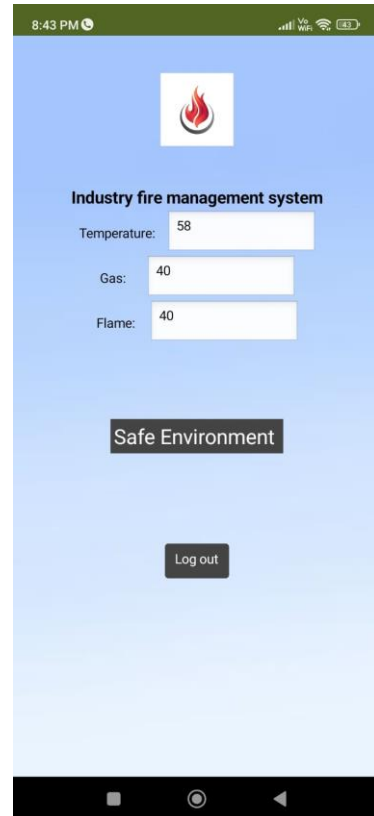
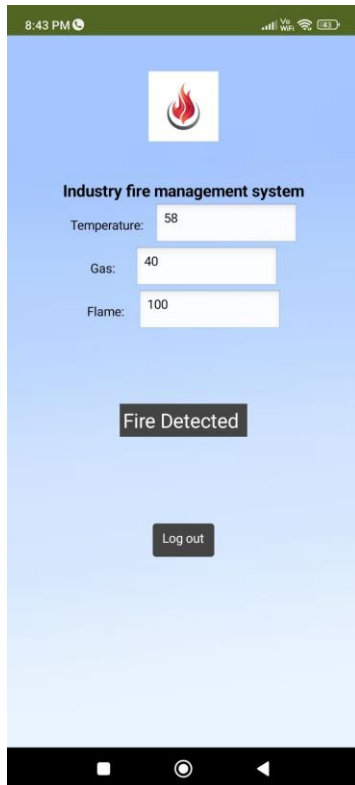


## SCREEN 3:



## MIT MOBILE APP OUTPUT:





If the values of flame is above 90 the fire is detected. If the values of the flame is below 90 it is safe environment.

