

Project Development

Phase

Sprint - III

Date	11 November 2022
Team ID	PNT2022TMID47980
Project Name	Industry-Specific Intelligent Fire Management System

LINK: <https://wokwi.com/projects/347685130732569171>

LINK: <https://wokwi.com/projects/348658884417684052>

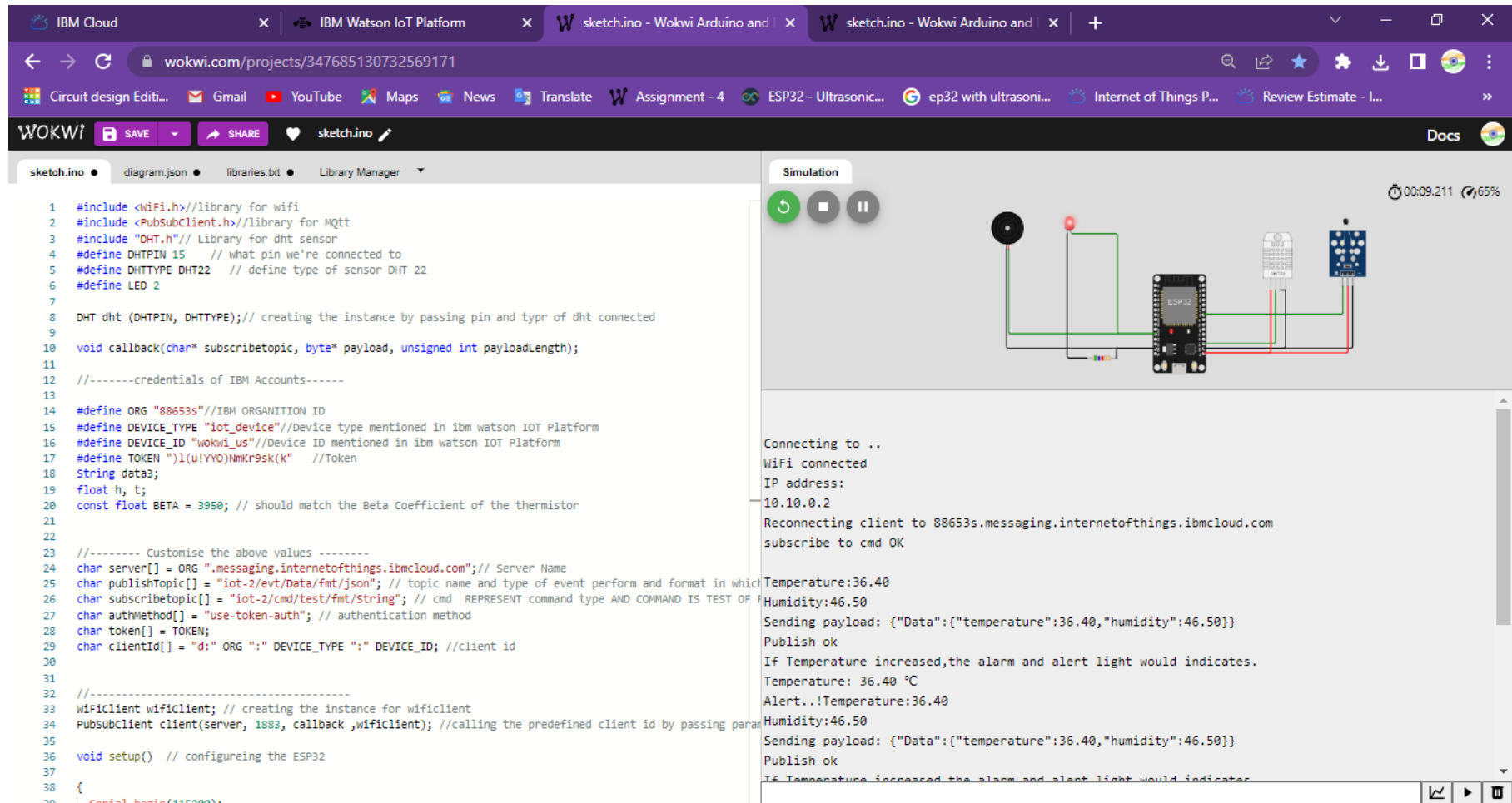
NODE-RED DASHBOARD UILINK:

<https://node-red-iwivz-2022-11-13.eu-gb.mybluemix.net/ui/#!/0?socketid=RNNTsORzKbrlp-UqAAAu>

WEB UI LINK : <https://node-red-dashboard059.eu-gb.mybluemix.net/fire>

OUTPUT:

WOKWI SIMULATOR



The screenshot displays the Wokwi simulator interface. The top navigation bar includes links to IBM Cloud, IBM Watson IoT Platform, and the current project 'sketch.ino - Wokwi Arduino and'. The main workspace is divided into three sections: a code editor on the left, a simulation window in the center, and a serial terminal on the right.

Code Editor: The sketch.ino file contains the following code:

```
1 #include <WiFi.h> //library for wifi
2 #include <PubSubClient.h> //library for MQTT
3 #include "DHT.h" // Library for dht sensor
4 #define DHTPIN 15 // what pin we're connected to
5 #define DHTTYPE DHT22 // define type of sensor DHT 22
6 #define LED 2
7
8 DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin and type of dht connected
9
10 void callback(char* topic, byte* payload, unsigned int payloadLength);
11
12 //-----credentials of IBM Accounts-----
13
14 #define ORG "88653s" //IBM ORGANIZATION ID
15 #define DEVICE_TYPE "iot_device" //Device type mentioned in IBM Watson IoT Platform
16 #define DEVICE_ID "wokwi_us" //Device ID mentioned in IBM Watson IoT Platform
17 #define TOKEN "1(u1YYO)NmKr9sk(k" //Token
18 String data3;
19 float h, t;
20 const float BETA = 3950; // should match the Beta Coefficient of the thermistor
21
22 //----- Customise the above values -----
23
24 char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name
25 char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform and format in which
26 char subscribeTopic[] = "iot-2/cmd/test/fmt/String"; // cmd REPRESENT command type AND COMMAND IS TEST OF
27 char authMethod[] = "use-token-auth"; // authentication method
28 char token[] = TOKEN;
29 char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID; //client id
30
31 //-----
32 WiFiClient wifiClient; // creating the instance for wifi client
33 PubSubClient client(server, 1883, callback, wifiClient); //calling the predefined client id by passing parameters
34
35 void setup() // configuring the ESP32
36 {
37   Serial.begin(115200);
38 }
```

Simulation Window: The simulation shows an ESP32 microcontroller connected to a DHT22 temperature and humidity sensor and an LED. The simulation is running at 65% speed.

Serial Terminal: The output of the simulation is as follows:

```
Connecting to ..
WiFi connected
IP address:
10.10.0.2
Reconnecting client to 88653s.messaging.internetofthings.ibmcloud.com
subscribe to cmd OK
Temperature:36.40
Humidity:46.50
Sending payload: {"Data":{"temperature":36.40,"humidity":46.50}}
Publish ok
If Temperature increased, the alarm and alert light would indicate.
Temperature: 36.40 °C
Alert..!Temperature:36.40
Humidity:46.50
Sending payload: {"Data":{"temperature":36.40,"humidity":46.50}}
Publish ok
If Temperature increased, the alarm and alert light would indicate.
```

OUTPUT:

WOKWI SIMULATOR

The screenshot displays the Wokwi simulator interface. On the left, a code editor shows the following C++ code for an ESP32-based fire alarm system:

```
1 #include <time.h>
2
3 bool exhaust_fan_on = false;
4 bool sprinkler_on = false;
5
6 float temperature = 0;
7 int gas = 0;
8 int flame = 0;
9
10 String flame_status = "";
11 String accident_status = "";
12 String sprinkler_status = "";
13
14 void setup() {
15   Serial.begin(99900);
16 }
17
18 void loop() {
19   //setting a random seed
20   srand(time(0));
21
22   //initial variable
23
24   temperature = random(-20,125);
25   gas = random(0,1000);
26   int flamereading = random(200,1024);
27   flame = map(flamereading,0,1024,0,2);
28
29   //set a flame status
30
31   switch (flame) {
32     case 0:
33       flame_status = "No Fire";
34       Serial.println("Flame Status : "+flame_status);
35       break;
```

On the right, the simulation window shows the status of the system at two different points in time, separated by a separator line:

Flame Status : No Fire
Gas Status : Gas leakage Detected
Sprinkler Status : not working
Exhaust fan Status : Working

-----/*****-----

Flame Status : Fire is Detected
Gas Status : Gas leakage Detected
Sprinkler Status : working
Exhaust fan Status : Working

-----/*****-----

IBM WATSON OUTPUT

IBM Cloud

IBM Watson IoT Platform

W sketch.ino - Wokwi Arduino and

W sketch.ino - Wokwi Arduino and

+

← → ↻

88653s.internetofthings.ibmcloud.com/dashboard/devices/browse

🔍 📄 ⭐ ⚙️ ⬇️ 🖨️ 🌐

Circuit design Editi...

Gmail

YouTube

Maps

News

Translate

Assignment - 4

ESP32 - Ultrasonic...

ep32 with ultrasoni...

Internet of Things P...

Review Estimate - I...

»

IBM Watson IoT Platform

tskarthicktskarthick6778@gmail.com
ID: 88653s

?

⬆️

⋮

🔌

👤

🏠

🌐

📶

🔧

⚙️

Browse

Action

Device Types

Interfaces

Add Device +

<input type="checkbox"/>	Device ID	Status	Device Type	Class ID	Date Added	Descriptive Location	Added By	Device Class	Firmware Version
> <input type="checkbox"/>	iot_device_1	● Connected	iot_device	Device	Nov 8, 2022 9:58 PM		tskarthicktskarthick6778@gmail.com		
> <input type="checkbox"/>	iot_device_2	● Connected	iot_device	Device	Nov 8, 2022 9:53 PM		tskarthicktskarthick6778@gmail.com		
> <input type="checkbox"/>	iot_device_3	● Connected	iot_device	Device	Nov 8, 2022 10:03 PM		tskarthicktskarthick6778@gmail.com		
▼ <input type="checkbox"/>	wokwi_us	● Connected	iot_device	Device	Nov 2, 2022 10:21 AM		tskarthicktskarthick6778@gmail.com		→ ...

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
Data	{"Data":{"temperature":36.4,"humidity":46.5}}	json	a few seconds ago
Data	{"Data":{"temperature":36.4,"humidity":46.5}}	json	19 minutes ago
Data	{"Data":{"temperature":36.4,"humidity":46.5}}	json	19 minutes ago
Data	{"Data":{"temperature":36.4,"humidity":46.5}}	json	19 minutes ago
Data	{"Data":{"temperature":36.4,"humidity":46.5}}	json	19 minutes ago

5 Simulations running

TRANSFERRING DATA FROM IBM WATSON INTO NODE-RED

The screenshot displays the Node-RED web interface in a browser. The main workspace, titled 'Flow 1', contains a flow starting with an 'IBM IoT' node (blue) which is connected. This node is wired to seven function nodes (orange) labeled 'temp', 'Gas', 'Flame', 'Fire Status', 'Sprinkler Status', 'Gas Status', and 'Exhaust Fan Status'. These function nodes are then connected to three output nodes (blue): 'Temperature' (with a line graph icon), 'Gas' (with a gauge icon), and 'Flame' (with a line graph icon). A 'msg.payload' node (green) is also connected to the function nodes. The left sidebar shows a list of available nodes including switch, numeric, text input, colour picker, date picker, slider, text, gauge, form, notification, audio out, ui control, chart, and template. The right sidebar, labeled 'debug', shows a log of messages received from the IoT node, including timestamps, node IDs, and the payload data. The bottom of the image shows the Windows taskbar with various application icons and the system clock indicating 00:17 on 15-11-2022.

Node-RED : node-red-iwivz-2022 x Node-RED Dashboard
node-red-iwivz-2022-11-13.eu-gb.mybluemix.net/red/#

Node-RED

Flow 1

filter nodes

switch

numeric

text input

colour picker

date picker

slider

text

gauge

form

notification

audio out

ui control

chart

template

IBM IoT

connected

temp

Gas

Flame

Fire Status

Sprinkler Status

Gas Status

Exhaust Fan Status

msg.payload

Temperature

Gas

Flame

debug

all nodes

all

msg.payload : number

670

11/15/2022, 12:15:39 AM node: f2f2649a.0d0d98
iot-2/type/NodeMCU/id/12345/evt/data/fmt/json :
msg.payload : number

332

11/15/2022, 12:15:40 AM node: f2f2649a.0d0d98
iot-2/type/NodeMCU/id/12345/evt/data/fmt/json :
msg.payload : string[7]

"No Fire"

11/15/2022, 12:15:41 AM node: f2f2649a.0d0d98
iot-2/type/NodeMCU/id/12345/evt/data/fmt/json :
msg.payload : string[11]

"Not Working"

11/15/2022, 12:15:42 AM node: f2f2649a.0d0d98
iot-2/type/NodeMCU/id/12345/evt/data/fmt/json :
msg.payload : string[23]

"Gas Leakage is Detected"

11/15/2022, 12:15:43 AM node: f2f2649a.0d0d98
iot-2/type/NodeMCU/id/12345/evt/data/fmt/json :
msg.payload : string[7]

"Working"

Type here to search

Screenshots

sketch.ino - Wokwi...

Node-RED : node-r...

00:17
15-11-2022

IBM App Development x Node-RED : node-red x node-red-contrib-so x IBM Watson IoT Platform x Running Node-RED lo x Node-RED x

127.0.0.1:1880/#flow/fea07489eb1f1f2b

Circuit design Edit... Gmail YouTube Maps News Translate Assignment - 4 ESP32 - Ultrasonic... ep32 with ultrasoni... Internet of Things P... Review Estimate - I...

Node-RED

Deploy

filter nodes

Flow 1 Flow 2 Flow 3

common

inject

debug

complete

catch

status

link in

link call

link out

comment

function

function

IBM IoT

connected

Temperature

Humidity

sensors

httpfunctionnode

http request

debug 3

Temperature

Humidity

debug

all nodes

all

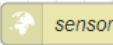
11/15/2022, 11:45:37 AM node: debug 3
iot-2/type/iot_device/id/wokwi_us/evt/Default/fmt/json :
msg.payload : Object
{ temp: 36.4, humid: 46.5, Alert...!:
"Alarm and Alert Light will be ..." }
11/15/2022, 11:45:39 AM node: debug 2
iot-2/type/iot_device/id/iot_device_2/evt/Default/fmt/json :
msg.payload : Object
{ temp: 36.4, humid: 46.5, Alert...!:
"Alarm and Alert Light will be ..." }
11/15/2022, 11:46:43 AM node: debug 3
iot-2/type/iot_device/id/wokwi_us/evt/Default/fmt/json :
msg.payload : Object
{ temp: 36.4, humid: 46.5, Alert...!:
"Alarm and Alert Light will be ..." }



connected

Temperature

Humidity



httpfunctionnode

http request

debug 3

Temperature

Humidity

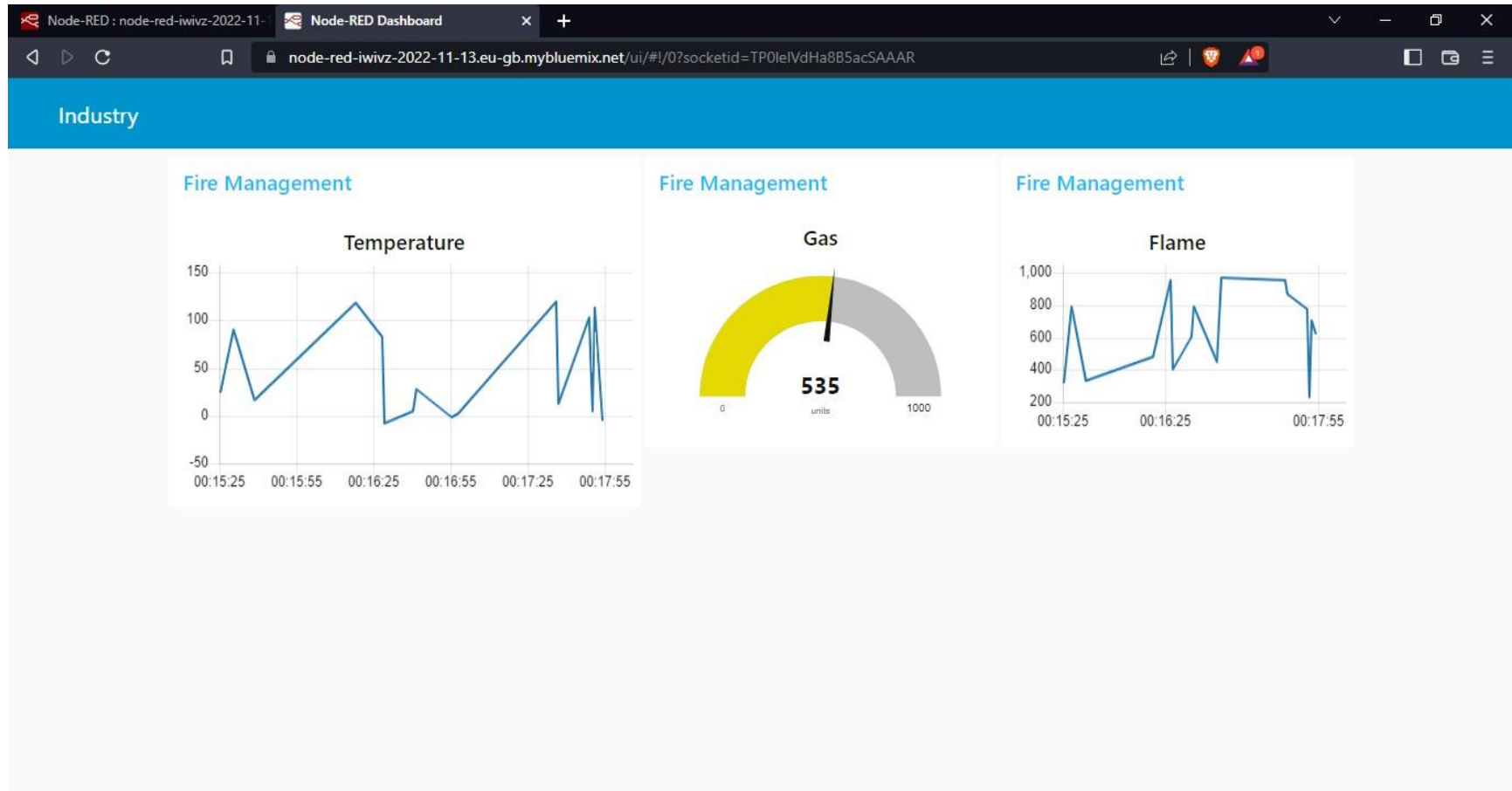
debug

all nodes

all

```
11/15/2022, 11:45:37 AM node: debug 3
iot-2/type/iot_device/id/wokwi_us/evt/Default/fmt/json :
msg.payload : Object
  { temp: 36.4, humid: 46.5, Alert...!:
    "Alarm and Alert Light will be ..." }
11/15/2022, 11:45:39 AM node: debug 2
iot-2/type/iot_device/id/iot_device_2/evt/Default/fmt/json :
msg.payload : Object
  { temp: 36.4, humid: 46.5, Alert...!:
    "Alarm and Alert Light will be ..." }
11/15/2022, 11:46:43 AM node: debug 3
iot-2/type/iot_device/id/wokwi_us/evt/Default/fmt/json :
msg.payload : Object
  { temp: 36.4, humid: 46.5, Alert...!:
    "Alarm and Alert Light will be ..." }
```

NODE DASHBOARD



Node-RED Dashboard

127.0.0.1:1880/ui/#/0?socketid=KCJrP7Z50nZJbVFFAAAG

Circuit design Editi... Gmail YouTube Maps News Translate Assignment - 4 ESP32 - Ultrasonic... ep32 with ultrasoni... Internet of Things P... Review Estimate - I...

Fire_Management

PNT2022TMID47980

Humidity

46.28

EXHAUST FAN ON

EXHAUST FAN OFF

Temperature

35.96

Water Sprinkler /ON

Humidity

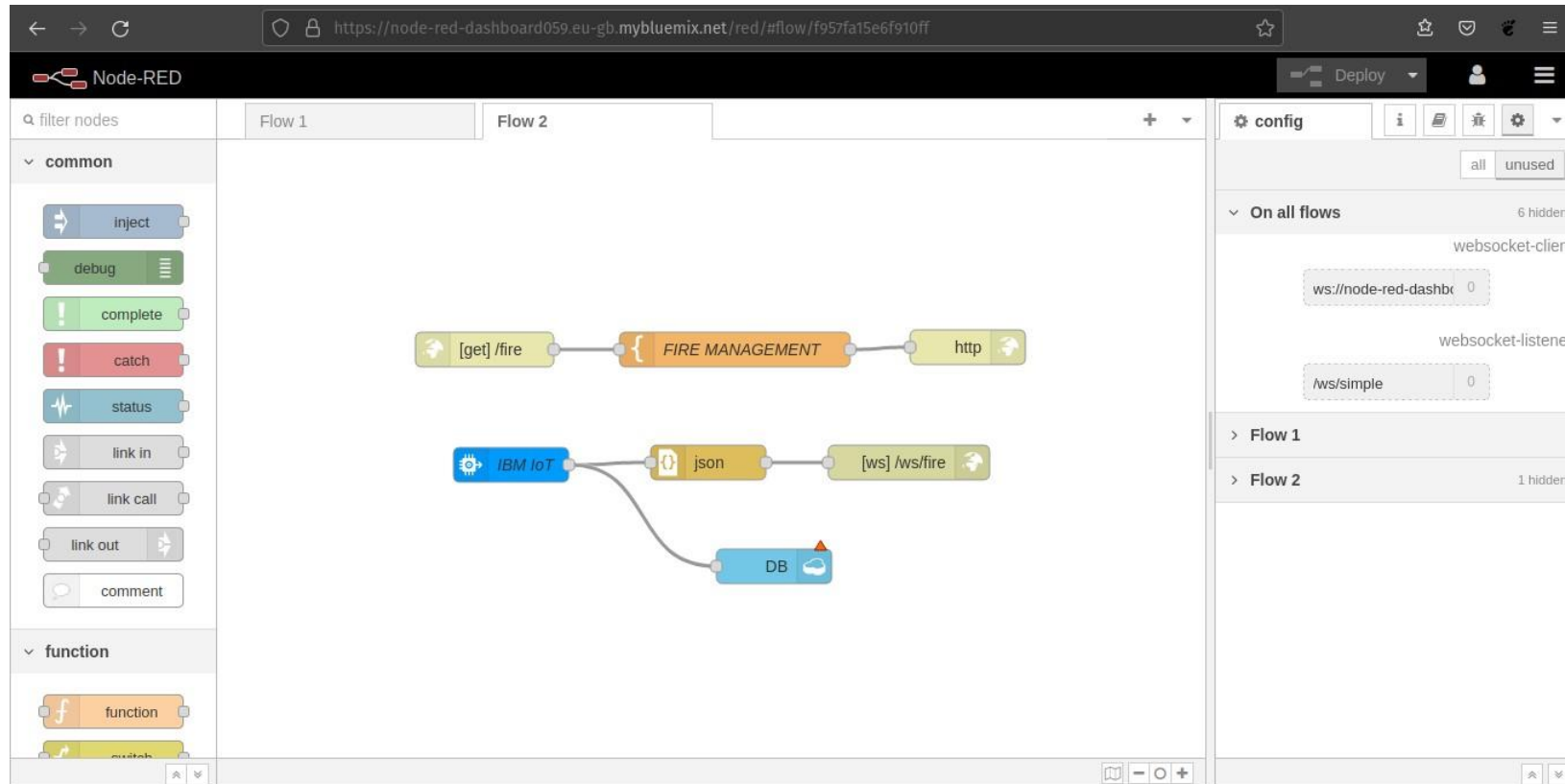
44.27%

Water Sprinkler/OFF

Temperature

35.96

TRANSFERRING DATA FROM NODE-RED INTO WEB UI

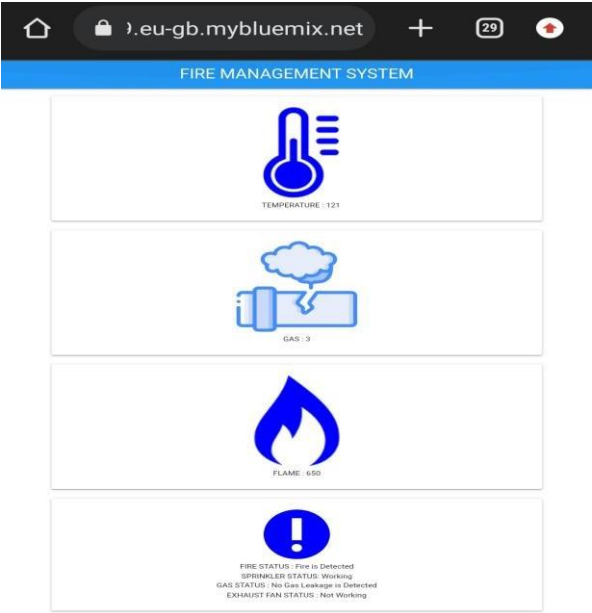


WEB UI

DESKTOP VIEW



MOBILE VIEW



↔

⏪ db ⋮

Document ID ▾

⚙️ Options

{ } JSON

📖

🔔

All Documents +

Query

Permissions

Changes

Design Documents +

☐

📄 Table

Metadata

{ } JSON

📄

Create Document

	id	key	value
☐ 📄	657846f21e0cb8ead462fd89321d28...	657846f21e0cb8ead462fd89321d28...	{ "rev": "1-1c9683229f242d4133b7f...
☐ 📄	657846f21e0cb8ead462fd89321dd3...	657846f21e0cb8ead462fd89321dd3...	{ "rev": "1-8aeef9d453a632f539ee9c...
☐ 📄	657846f21e0cb8ead462fd8932201e...	657846f21e0cb8ead462fd8932201e...	{ "rev": "1-7b6df30912cf9fde43ca8b...
☐ 📄	657846f21e0cb8ead462fd8932203d...	657846f21e0cb8ead462fd8932203d...	{ "rev": "1-a9bec25d7f94ccc71ce692...
☐ 📄	70ea2e4bb2a9c635be3ce2603a25a...	70ea2e4bb2a9c635be3ce2603a25a...	{ "rev": "1-b567b4cce122c31e1666fc...
☐ 📄	70ea2e4bb2a9c635be3ce2603a268...	70ea2e4bb2a9c635be3ce2603a268...	{ "rev": "1-217497b95c16c3d228800...
☐ 📄	70ea2e4bb2a9c635be3ce2603a272...	70ea2e4bb2a9c635be3ce2603a272...	{ "rev": "1-a01738b27517a2bb4b93b...
☐ 📄	70ea2e4bb2a9c635be3ce2603a273...	70ea2e4bb2a9c635be3ce2603a273...	{ "rev": "1-13230a9f364a021a02422...
☐ 📄	7170def319e06e12e85b74c728897...	7170def319e06e12e85b74c728897...	{ "rev": "1-4bdfcbf4dbbf888784fc24d...
☐ 📄	7170def319e06e12e85b74c7288b7...	7170def319e06e12e85b74c7288b7...	{ "rev": "1-5b1a46d23a6c259bd5b97...
☐ 📄	7170def319e06e12e85b74c7288c2...	7170def319e06e12e85b74c7288c2...	{ "rev": "1-783ab5b4e98aed22641a1...

Showing document 1 - 20.

Documents per page: 20 ▾

⏪ ⏩

Log Out

Log Out

✓ Save Changes

Cancel

Upload Attachment

Clone Document

Delete

1

```
"_id": "657846f21e0cb8ead462fd89321d28fd",
2
3  "_rev": "1-1c9683229f242d4133b7fae068107c43",
4  "gas": 267,
5  "temperature": 50,
6  "flame": 931,
7  "fire_status": "Fire is Detected",
8  "sprinkler_status": "Working",
9  "Gas_status": "Gas Leakage is Detected",
10 "exhaust_fan_status": "Working"
11
```

CODE:

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

#define ORG "88653s"
#define DEVICE_TYPE "iot_device"
#define DEVICE_ID "wokwi_us"
#define TOKEN ")l(u!YYO)NmKr9sk(k"

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";
    }
```



```
}
```

```
//Wokwi Project
```

```
#include <WiFi.h>//library for wifi
#include <PubSubClient.h>//library for MQTT
#include "DHT.h"// Library for dht sensor
#define DHTPIN 15    // what pin we're connected to
#define DHTTYPE DHT22 // define type of sensor DHT 22
#define LED 14

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 "88653s"//IBM ORGANITION ID
#define DEVICE_TYPE "iot_device"//Device type mentioned in ibm watson IOT
Platform
#define DEVICE_ID "wokwi_us"//Device ID mentioned in ibm watson IOT Platform
#define TOKEN ")l(u!YY0)NmKr9sk(k" //Token
String data3;
float h, t;
```

```

const float BETA = 3950; // should match the Beta Coefficient of the thermistor

//----- 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/test/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();
  delay(10);
  Serial.println();
  wificonnect();
  mqttconnect();
}

```

```

    Serial.begin(9600);
    analogReadResolution(10);
    pinMode(18, INPUT);
    pinMode(14, OUTPUT);
    pinMode(12, OUTPUT);
}

void loop() // Recursive Function
{

    h = dht.readHumidity();
    t = dht.readTemperature();
    Serial.print("Temperature:");
    Serial.println(t);
    Serial.print("Humidity:");
    Serial.println(h);

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

    //.....Analog Temperature Sensor.....

    int analogValue = analogRead(18);

```

```

    float celsius = 1 / (log(1 / (1023. / analogValue - 1)) / BETA + 1.0 / 298.15)
+ 36.4;
    Serial.print("Temperature: ");
    Serial.print(celsius);
    Serial.println(" °C");
    Serial.print("Alert..!");

    if(celsius >= 35)
        digitalWrite(14, HIGH);
    else
        digitalWrite(14, LOW);
    delay(1000);

}

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

void PublishData(float temp, float humid) {
    mqttconnect(); //function call for connecting to ibm

    /*
        creating the String in in form JSON to update the data to ibm cloud
    */

    String payload = "{\"Data\":{\"temperature\":";
    payload += temp;
    payload += ", \"humidity\":";

```

```
payload += humid;  
payload += "}}";
```

```
Serial.print("Sending payload: ");  
Serial.println(payload);
```

```
if (client.publish(publishTopic, (char*) payload.c_str())) {  
    Serial.println("Publish ok"); // if it successfully upload data on the cloud  
then it will print publish ok in Serial monitor or else it will print publish  
failed  
    Serial.println("If Temperature increased,the alarm and alert light would  
indicates. ");  
} 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="";
}

```

```
}
```

//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_status+"\""}";
```

```
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();
  }
}
```

//.....Project Data in json Format...../

```
{
  "version": 1,
  "author": "T S Karthick",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 10, "left": -60.67, "attrs": {} },
    {
      "type": "wokwi-led",
      "id": "led1",
      "top": -109,
      "left": -244.4,
      "attrs": { "color": "red" }
    },
  ],
}
```

```
{
  "type": "wokwi-dht22",
  "id": "dht1",
  "top": -70.9,
  "left": 157.2,
  "attrs": { "temperature": "36.4", "humidity": "46.5" }
},
{
  "type": "wokwi-ntc-temperature-sensor",
  "id": "ntc1",
  "top": -69.55,
  "left": 253.55,
  "rotate": 90,
  "attrs": {}
},
{
  "type": "wokwi-resistor",
  "id": "r1",
  "top": 169.5,
  "left": -190.59,
  "attrs": { "value": "5600" }
},
{
  "type": "wokwi-buzzer",
  "id": "bz1",
  "top": -118.83,
  "left": -378.64,
  "attrs": { "volume": "0.1" }
}
],
```

```

"connections": [
  [ "esp:TX0", "$serialMonitor:RX", "", [] ],
  [ "esp:RX0", "$serialMonitor:TX", "", [] ],
  [ "dht1:GND", "esp:GND.1", "black", [ "v0" ] ],
  [ "dht1:SDA", "esp:D15", "green", [ "v0" ] ],
  [ "ntc1:GND", "esp:GND.1", "black", [ "v0" ] ],
  [ "ntc1:VCC", "esp:3V3", "red", [ "v0" ] ],
  [ "led1:C", "r1:1", "black", [ "v0" ] ],
  [ "r1:2", "esp:GND.2", "black", [ "v0" ] ],
  [ "led1:A", "esp:D14", "green", [ "v-0.86", "h89.56", "v199.46" ] ],
  [ "ntc1:OUT", "esp:D18", "green", [ "v0" ] ],
  [ "bz1:1", "esp:GND.2", "black", [ "v0" ] ],
  [ "bz1:2", "esp:D14", "green", [ "v0" ] ],
  [ "dht1:VCC", "esp:3V3", "red", [ "v0" ] ],
  [ "dht1:NC", "dht1:GND", "black", [ "v0" ] ]
]
}

```

//.....Python Script for Random Outputs of Temperature and Humidity.....

```

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

```

```

#Provide your IBM Watson Device Credentials
organization = "bxobbs"

```

```
deviceType = "b5ibm"  
deviceId = "b5device"  
authMethod = "token"  
authToken = "b55m1eibm"
```

```
# Initialize GPIO
```

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

```
try:  
    deviceOptions = {"org": organization, "type": deviceType, "id": deviceId, "auth-method": authMethod, "auth-  
token": authToken}  
    deviceCli = ibmiotf.device.Client(deviceOptions)  
    #.....  
  
except Exception as e:  
    print("Caught exception connecting device: %s" % str(e))  
    sys.exit()
```

```
# Connect and send a datapoint "hello" with value "world" into the cloud as an event of type "greeting" 10 times
deviceCli.connect()
```

```
while True:
```

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

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

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

```
    data = { 'temp' : temp, 'Humid': Humid }
```

```
    #print data
```

```
    def myOnPublishCallback():
```

```
        print ("Published Temperature = %s C" % temp, "Humidity = %s %" % Humid, "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
```

```
# Disconnect the device and application from the cloud
```

```
deviceCli.disconnect()
```