

## SPRINT 4

Team ID	PNT2022TMID37599
Project Name	Hazardous Area Monitoring for Industrial Plant powered by IoT
Team ID	Chanukya.K(TL) Anuhya.k Jyothsna.J Silparani.Y Swetha.J

### WOKWI CODE:

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

DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin and typr of dht
connected
void callback(char* subscribetopic, byte* payload, unsigned int
payloadLength);
//-----credentials of IBM Accounts-----
#define ORG "iagqzu" //IBM ORGANITION ID
#define DEVICE_TYPE "Deepak" //Device type mentioned in ibm watson IOT Platform
#define DEVICE_ID "123" //Device ID mentioned in ibm watson IOT Platform
#define TOKEN "12345678"
//Token String data3; float h,
t;

//----- Customise the above values -----
char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name char
publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform
and format in which data to be send
char subscribetopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT command type
AND COMMAND IS TEST OF FORMAT STRING char authMethod[] = "use-token-auth"; //
authentication method char token[] = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID; //client id

// -----
WiFiClient wifiClient; // creating the instance for wificlient
PubSubClient client(server, 1883, callback ,wifiClient); //calling the predefined
client id by passing parameter like server id,portand wificredential
```

```
void setup()// configureing the  
ESP32
```

```
{  
  Serial.begin(115200);  
  dht.begin();  
  pinMode(LED,OUTPUT);  
  delay(10);  
  Serial.println();  
  wificonnect();  
  mqttconnect();  
}
```

```
void loop()// Recursive  
Function
```

```
{  
  
  h = dht.readHumidity(); t  
  = dht.readTemperature();  
  Serial.print("temp:");  
  Serial.println(t);  
  Serial.print("Humid:");  
  Serial.println(h);  
  PublishData(t, h);  
  delay(1000); if  
  (!client.loop()) {  
    mqttconnect();  
  }  
}
```

```
/*.....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 =  
  "{\"temp\""; payload +=  
  temp; payload += ","  
  "\"Humid\""; payload +=  
  humid; payload += "}";  
  
  Serial.print("Sending payload: ");  
  Serial.println(payload);
```

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

```

    } else {
        Serial.println("Publish failed");
    }
}

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

    WiFi.begin("Wokwi-GUEST", "", 6); //passing the wifi credentials to establish the
connection
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print("."); }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}
void initManagedDevice()
{
    if (client.subscribe(subscribetopic)) {
        Serial.println((subscribetopic));
        Serial.println("subscribe to cmd OK");
    } else {
        Serial.println("subscribe to cmd FAILED");
    }
}
void callback(char* subscribetopic, byte* payload, unsigned int
payloadLength)
{
    Serial.print("callback invoked for topic: ");
    Serial.println(subscribetopic);
    for (int i = 0; i < payloadLength; i++)
        { //Serial.print((char)payload[i]);
          data3 += (char)payload[i];
        }
}

```

```

Serial.println("data: "+ data3);
if(data3=="lighton") {
Serial.println(data3);
digitalWrite(LED,HIGH); } else
{
Serial.println(data3);
digitalWrite(LED,LOW);
} data3="";
}

```

## WOKWI OUTPUT:

The screenshot displays the WOKWI IDE interface. On the left, 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 DHT11
4 #define DHTPIN 15 // what pin we're connected to
5 #define DHTTYPE DHT22 // define type of sensor DHT 11
6 #define LED 3
7
8 DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin and type of dht correct
9
10 void callback(char* topic, byte* payload, unsigned int payloadLength);
11
12 //----- credentials of AWS accounts -----
13
14 #define AWS "13099" //AWS ORGANIZATION ID
15 #define DEVICE_TYPE "awsiot" //Device type mentioned in the Watson IoT Platform
16 #define DEVICE_ID "1234" //Device ID mentioned in the Watson IoT Platform
17 #define TOKEN "12345678" //Token
18 String data1;
19 float h, t;
20
21 //----- (initialize the above values) -----
22
23 char server[] = AWS ".messaging.internetofthings.ibmcloud.iot"; // server name
24 char publishTopic[] = "iot-2/v1/data/iot/1234"; // topic name and type of event perform
25 char subscribeTopic[] = "iot-2/cmd/command/1234/string"; // cmd. RESPONSE command type AWS
26 char authMethod[] = "use-token-auth"; // authentication method
27 char token[] = TOKEN;
28 char clientId[] = "0:" AWS ":" DEVICE_TYPE ":" DEVICE_ID; //client id
29
30 //
31
32 WiFiClient wifiClient; // creating the instance for wifiClient
33 PubSubClient client(server, 1883, callback, wifiClient); //calling the predefined client

```

On the right, the 'Simulation' window shows a visual representation of the hardware: an ESP32 microcontroller, a red LED, and a DHT22 temperature and humidity sensor. Below the simulation, the console output shows the following sequence of events:

```

Humid:48.00
Sending payload: {"temp":24.00,"Humid":48.00}
Publish ok
temp:24.00
Humid:48.00
Sending payload: {"temp":24.00,"Humid":48.00}
Publish ok

```

## IBM WATSON PLATFORM ⑨

## DEVICE EVENT LOG:

The screenshot shows a web application for managing IoT devices. At the top, there are tabs for 'Browse', 'Actions', 'Device Types', and 'Interfaces'. A search bar is labeled 'Search by Device ID'. Below this is a table of devices with columns: Device ID, Status, Device Type, Class ID, Date Added, and Descriptive Location. One device is listed with ID '123', status 'Connected', type 'Onepuls', class 'Device', and date 'Sep 29, 2022 7:54 PM'. Below the table, there are tabs for 'Identity', 'Device Information', 'Recent Events', 'State', and 'Logs'. The 'Recent Events' tab is active, showing a list of events with columns: Event, Value, Format, and Last Received. The events are JSON payloads from 'IoTService' containing temperature and humidity data.

Device ID	Status	Device Type	Class ID	Date Added	Descriptive Location
123	Connected	Onepuls	Device	Sep 29, 2022 7:54 PM	

  

Event	Value	Format	Last Received
IoTService	{"temp":32,"humid":33}	json	a few seconds ago
IoTService	{"temp":34,"humid":87}	json	a few seconds ago
IoTService	{"temp":35,"humid":6}	json	a few seconds ago
IoTService	{"temp":86,"humid":24}	json	a few seconds ago

## DEVICE EVENT PAYLOAD:

The screenshot shows a modal window titled 'Event Payload' overlaid on the device management dashboard. The modal displays the event name 'IoTService' and the time received 'Nov 04, 2022 11:31:49'. Below this is a JSON payload: {"temp": 32, "humid": 33}.

Event Name	Time Received
IoTService	Nov 04, 2022 11:31:49

  

```

{
  "temp": 32,
  "humid": 33
}
  
```

## DEVICE- BOARD:

