Final code

Date	17 November 2022
Team ID	PNT2022TMID47379
Project Name	IOT BASED CROP PROTECTION SYSTEM FOR AGRICULTURE

Source code using Python :

```
import random import
ibmiotf.application import
ibmiotf.device from time import sleep
import sys


#IBM Watson Device Credentials.
organization = "l1a82f" deviceType
= "cibie" deviceId
= "cibie123" authMethod =
"token" authToken =
"12345678"


def myCommandCallback(cmd):     print("Command received:
%s" % cmd.data['command'])     status = cmd.data['command']
if status== "sprinkler_on":
print ("sprinkler is ON")     else :          print
("sprinkler 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()


#Connecting to IBM watson.
deviceCli.connect()


while True:
    temp_sensor = round( random.uniform(0,80),2)     PH_sensor = round(random.uniform(1,14),3)     camera  =
["Detected","Not Detected","Not Detected","Not Detected","Not Detected","Not Detected",]     camera_reading
```

```
= random.choice(camera)          flame = ["Detected","Not Detected","Not Detected","Not Detected","Not Detected","Not Detected",]
flame_reading = random.choice(flame)    moist_level = round(random.uniform(0,100),2)    water_level = round(random.uniform(0,30),2)


#storing the sensor data to send in json format to cloud.


   temp_data = { 'Temperature' : temp_sensor }          PH_data
= { 'PHLevel' : PH_sensor }
camera_data = { 'Animal attack' : camera_reading}     flame_data
= { 'Flame' : flame_reading }    moist_data
= { 'Moisture Level' : moist_level}    water_data = {
'Water Level' : water_level}


# publishing Sensor data to IBM Watson for every 5-10 seconds.
   success = deviceCli.publishEvent("Temperature sensor", "json", temp_data, qos=0)       sleep(1)
if success:
      print ("   ..... .....publis h ok.....   ")
      print ("Published Temperature = %s C" % temp_sensor,  "to IBM Watson")    success =
deviceCli.publishEvent("PH sensor", "json", PH_data, qos=0)      sleep(1)    if  success:
      print ("Published PHLevel = %s" % PH_sensor,  "to IBM Watson")            success
= deviceCli.publishEvent("camera", "json", camera_data, qos=0)      sleep(1)    if
success:
      print ("Published Animal attack %s " % camera_reading,  "to IBM Watson")       success
= deviceCli.publishEvent("Flame sensor", "json", flame_data, qos=0)      sleep(1)    if
success:
      print ("Published Flame %s " % flame_reading,  "to IBM Watson")            success
= deviceCli.publishEvent("Moisture sensor", "json", moist_data, qos=0)      sleep(1)    if
success:
      print ("Published Moisture Level = %s " % moist_level,  "to IBM Watson")
success = deviceCli.publishEvent("Water sensor", "json", water_data, qos=0)      sleep(1)    if
success:
      print ("Published Water Level = %s cm" % water_level,  "to IBM Watson")          print
("")


#Automation to control sprinklers by present temperature an to send alert message to IBM Watson.
if (temp_sensor > 35):


      print("sprinkler-1 is ON")
      success = deviceCli.publishEvent("Alert1", "json",{ 'alert1' : "Temperature(%s) is high, sprinkerlers are turned ON" %temp_sensor }  ,  qos=0)
sleep(1)      if success:          print( 'Published alert1 : ',  "Temperature(%s) is high, sprinkerlers are turned ON" %temp_sensor,"to IBM Watson")
print("")     else:
      print("sprinkler-1 is OFF")
      print("")
#To send alert message if farmer uses the unsafe fertilizer to crops.
if (PH_sensor > 7.5 or PH_sensor < 5.5):
      success = deviceCli.publishEvent("Alert2", "json",{ 'alert2' : "Fertilizer PH level(%s) is not safe,use other fertilizer" %PH_sensor } ,
qos=0)          sleep(1)
if success:          print('Published alert2 : ' , "Fertilizer PH level(%s) is not safe,use other fertilizer" %PH_sensor,"to
IBM Watson")
print("")



#To send alert message to farmer that animal attack on crops.


   if (camera_reading == "Detected"):
      success = deviceCli.publishEvent("Alert3", "json", { 'alert3' : "Animal attack on crops detected" }, qos=0)          sleep(1)
if success:
```

```
        print('Published alert3 : ' , "Animal attack on crops detected","to IBM Watson","to IBM Watson")          print("")


#To send alert message if flame detected on crop land and turn ON the splinkers to take immediate action.


    if (flame_reading == "Detected"):        print("sprinkler-2 is ON")     success = deviceCli.publishEvent("Alert4", "json", { 'alert4' :
"Flame is detected crops are in danger,sprinklers turned ON" }, qos=0)       sleep(1)     if success:
        print( 'Published alert4 : ' , "Flame is detected crops are in danger,sprinklers turned ON","to IBM Watson")          print("")
else:
        print("sprinkler-2 is OFF")         print("")



#To send alert message if Moisture level is LOW and to Turn ON Motor-1 for irrigation.


    if (moist_level < 20):          print("Motor-1 is ON")     success = deviceCli.publishEvent("Alert5", "json", { 'alert5' : "Moisture
level(%s) is low, Irrigation started" %moist_level }, qos=0)       sleep(1)              if success:
        print('Published alert5 : ' , "Moisture level(%s) is low, Irrigation started" %moist_level,"to IBM Watson" )          print("")
else:
        print("Motor-1 is OFF")         print("")



     #To send alert message if Water level is HIGH and to Turn ON Motor-2 to take water out.


    if (water_level > 20):          print("Motor-2 is  ON")       success = deviceCli.publishEvent("Alert6", "json", { 'alert6' : "Water level(%s)
is high, so motor is ON to  take water out "  %water_level
}, qos=0)
    sleep(1)
if success:          print('Published alert6 : ' , "water level(%s) is high, so motor is ON to take water out " %water_level,"to
IBM Watson" )         print("")     else:
        print("Motor-2 of OFF")          print("")


    deviceCli.commandCallback = myCommandCallback # Disconnect
the device and application from the cloud deviceCli.disconnect()
```

Wowki code (Test case) :

```
//UV - Sensor const int trigPin

= 12; const int echoPin = 14;
```

```
//define sound velocity in cm/uS
#define SOUND_VELOCITY 0.034
#define CM_TO_INCH 0.393701
 long duration; int distanceCm; float
distanceInch; const char f[14]="flamedetected";
const char reg[8]="notfire"; const char
ani[15]="animaldetected"; const char
an[5]="safe";

//defining DHT22 sensor
#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
//Fire - Sensor int Buzzer =
32;       //int Fire_analog = 4;
int Fire_digital = 2;
//int Buzzer = 32;        // used for ESP32
//int Fire_analog = 4;    // used for ESP32
//int Fire_digital = 2;   // used for ESP32
// These constants should match the photoresistor's "gamma" and "rl10" attributes
#define LIGHT_SENSOR_PIN 34


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 "l1a82f"//IBM ORGANITION ID
#define DEVICE_TYPE "cibie"//Device type mentioned in ibm watson IOT Platform
#define DEVICE_ID "cibie123"//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  {
```

```
  //DHT22 - Sensor
Serial.begin(115200);
dht.begin();   delay(10);
Serial.println();   wificonnect();
mqttconnect();

//UV-Sensor
  Serial.begin(115200); // Starts the serial communication   pinMode(trigPin,
OUTPUT); // Sets the trigPin as an Output   pinMode(echoPin, INPUT); // Sets
the echoPin as an Input   delay(10);   Serial.println();   wificonnect();
mqttconnect();

 // Fire-sensor  Serial.begin(115200);
pinMode(Buzzer, OUTPUT);
pinMode(Fire_digital, INPUT);
delay(10);   Serial.println();
wificonnect();   mqttconnect();

}
void loop()// Recursive Function
{

//DHT22 - Sensor   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();
  }


  // UV - Sensor   // Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);   delayMicroseconds(10);
digitalWrite(trigPin, LOW);

  // Reads the echoPin, returns the sound wave travel time in microseconds   duration = pulseIn(echoPin, HIGH);

  // Calculate the distance   distanceCm = duration *
SOUND_VELOCITY/2;
```

```
  // Convert to inches   distanceInch = distanceCm *
CM_TO_INCH;

  // Prints the distance on the Serial Monitor
  Serial.print("Distance (cm): ");
  Serial.println(distanceCm);
  Serial.print("Distance (inch): ");   Serial.println(distanceInch);
if(distanceCm<175)
  {    digitalWrite (Buzzer, HIGH) ; //send tone
delay(1000);
    digitalWrite (Buzzer, LOW) ;
  }
  PublishData(distanceCm, distanceCm);
delay(1000);   if (!client.loop()) {     mqttconnect();
  }


  // reads the input on analog pin (value between 0 and 4095)   int analog =
analogRead(LIGHT_SENSOR_PIN);   int digital = digitalRead(Fire_digital);
  Serial.print("flame:");
  Serial.println(analog);
  Serial.print("flame:");
  Serial.println(digital);
  // We'll have a few threshholds, qualitatively determined    Data(analog,digital);
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");
```

```
  }

}
String payload; void PublishData(int distanceCm  , int distanceInch  ) {
mqttconnect();//function call for connecting to ibm
  /*      creating the String in in form JSon to update the data to ibm cloud
  */   if (distanceCm<175){    payload =
"{\"animalattack\":";
  //payload += duration;
  //payload += "," "\"distanceCm\":";    payload +=
ani;    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 Data(int analog  , int digital  ) {   mqttconnect();//function call
for connecting to ibm
  /*
     creating the String in in form JSon to update the data to ibm cloud   */
  String payload;   if
(analog>200){
   payload = "{\"flame\":";
  //payload += duration;
  //payload += "," "\"distanceCm\":";    payload +=
f;    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];
  }

}
```