

TEAM ID	PNT2022TMID27944
PROJECT NAME	IoT Based Smart Crop Protection System for Agriculture

Using Wokwi Platform

Source 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 "bx1po5"//IBM ORGANITION ID
#define DEVICE_TYPE "abcd"//Device type mentioned in ibm watson IOT Platform
#define DEVICE_ID "1234"//Device ID mentioned in ibm watson IOT Platform
#define TOKEN "12345678" //Token
String data3;

float h , t;
float m,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
{

  h = dht1.readHumidity();
  t = dht1.readTemperature();
  m = dht2.readHumidity();

  Serial.print("temp:");
  Serial.println(t);
  Serial.print("Humid:");
  Serial.println(h);
  Serial.print("moisture:");
  Serial.println(m);
  if(m<=50)
  {
    Serial.print("Low moisture , Motor 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,h,m);
delay(1000);
if (!client.loop()) {
    mqttconnect();
}
}

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

void PublishData(float temp, float Humid, int moisture) {
    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 += "," "\"moisture\":";
    payload += moisture;
    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 successfully 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 definition 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)
{
}

```

Output:

When Moisture is High – Motor OFF

The screenshot displays the Wokwi IoT simulator 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
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 dht connect
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 "bxipo5" //IBM ORGANIZATION ID
19 #define DEVICE_TYPE "abcd" //Device type mentioned in ibm watson IOT Platform
20 #define DEVICE_ID "1234" //Device ID mentioned in ibm watson IOT Platform
21 #define TOKEN "12345678" //Token
22 String data3;
23
24 float h , t;
25 float m,T;
26 int f;
27 int pos;
28
29
30 //----- Customise the above values -----
31 char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name
32 char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform at
33 char subscribetopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT command type AND
34 char authMethod[] = "use-token-auth"; // authentication method
35 char token[] = TOKEN;
36 char clientID[] = "dt" ORG "." DEVICE_TYPE "." DEVICE_ID "/" client ID

```

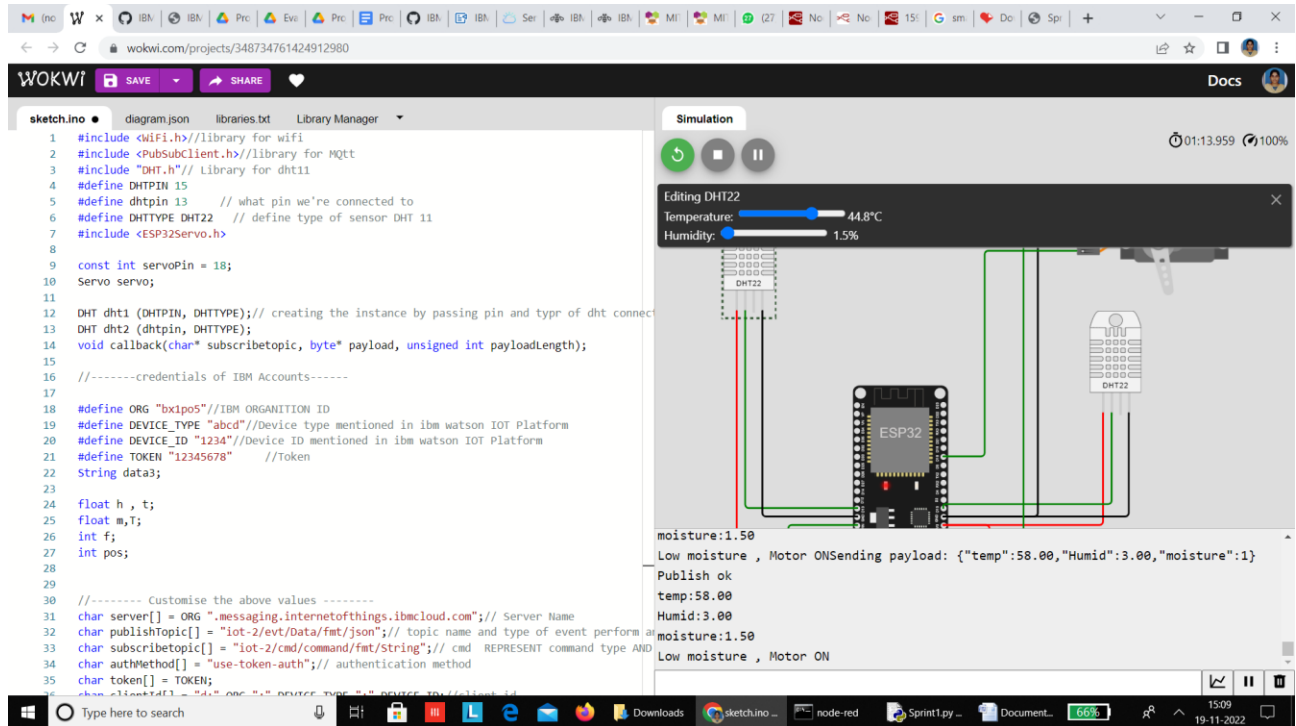
On the right, the 'Simulation' window shows a visual representation of the hardware: an ESP32 microcontroller, two DHT22 sensors, and a servo motor. The output console at the bottom shows the following sequence of events:

```

Sending payload: {"temp":58.00,"Humid":3.00,"moisture":90}
Publish ok
temp:58.00
Humid:3.00
moisture:90.50
Sending payload: {"temp":58.00,"Humid":3.00,"moisture":90}
Publish ok

```

When Moisture is Low – Motor ON:



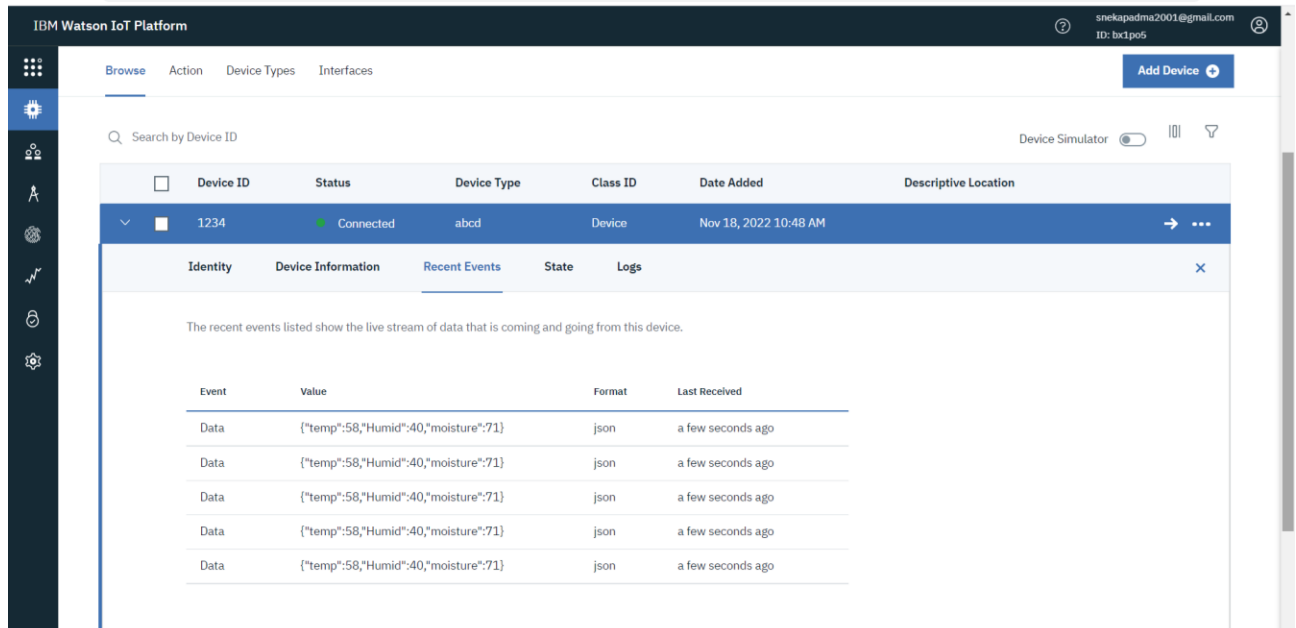
The screenshot displays the Wokwi IoT Platform interface. On the left, the sketch.ino file is open, showing the following code:

```
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
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 dht connect
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 "bx1po5" //IBM ORGANIZATION ID
19 #define DEVICE_TYPE "abcd" //Device type mentioned in ibm watson IOT Platform
20 #define DEVICE_ID "1234" //Device ID mentioned in ibm watson IOT Platform
21 #define TOKEN "12345678" //Token
22 String data3;
23
24 float h , t;
25 float m,T;
26 int f;
27 int pos;
28
29 //----- Customise the above values -----
30
31 char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name
32 char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform
33 char subscribetopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT command type AND
34 char authMethod[] = "use-token-auth"; // authentication method
35 char token[] = TOKEN;
36 char clientId[] = "44" ORG ".-". DEVICE_TYPE ".-". DEVICE_ID ".-". DEVICE_NAME ".-". DEVICE_ID
```

The simulation window on the right shows the DHT22 sensor reading 44.8°C and 1.5% humidity. The console output shows the following messages:

```
moisture:1.50
Low moisture , Motor ON
Sending payload: {"temp":58.00,"Humid":3.00,"moisture":1}
Publish ok
temp:58.00
Humid:3.00
moisture:1.50
Low moisture , Motor ON
```

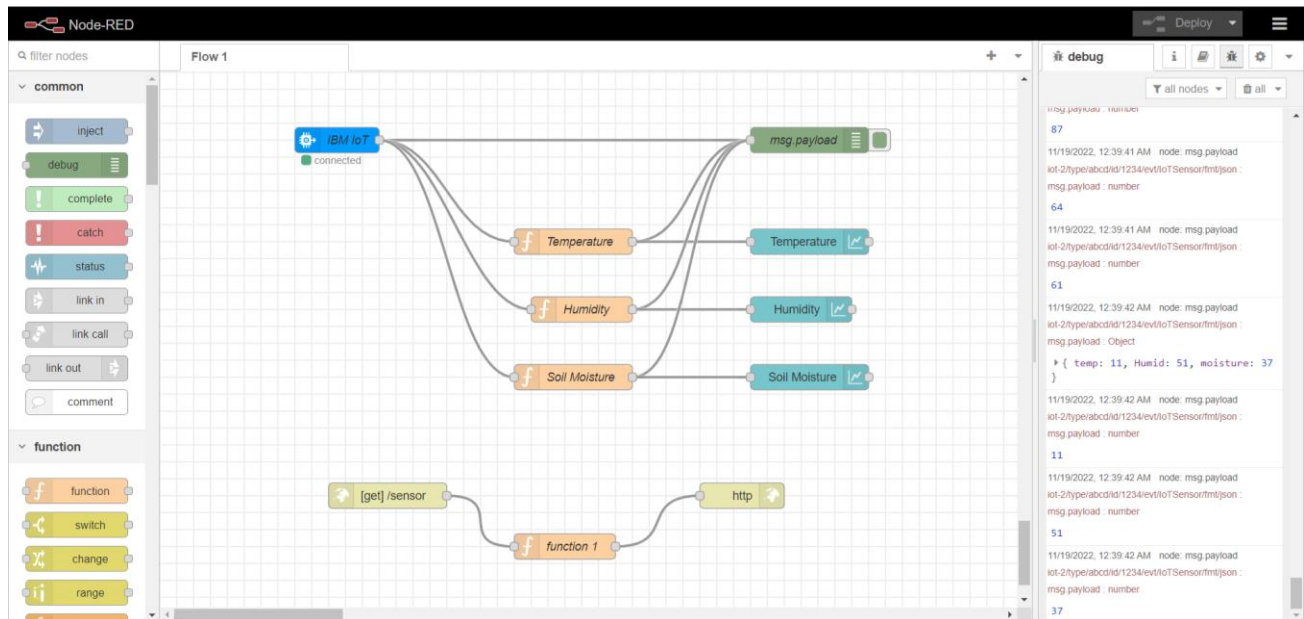
Wokwi to IBM Watson IoT Platform:



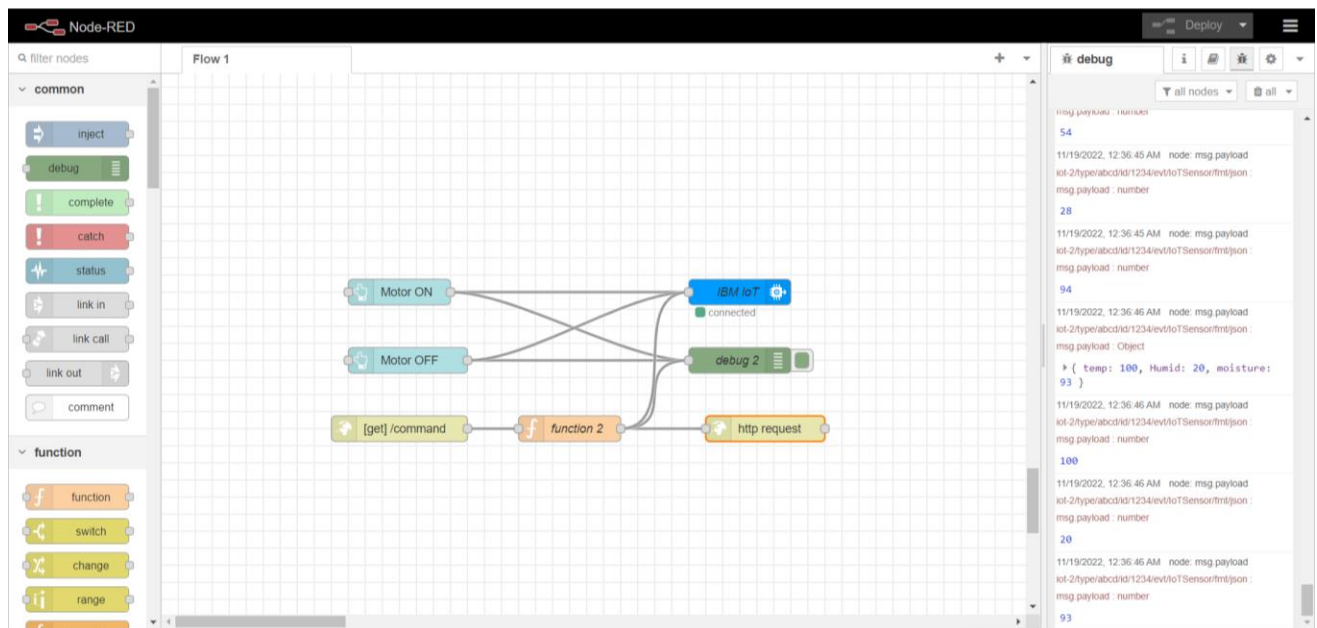
The screenshot displays the IBM Watson IoT Platform dashboard. The device with ID 1234 is shown as connected. The 'Recent Events' tab is selected, displaying a stream of data events.

Event	Value	Format	Last Received
Data	{"temp":58,"Humid":40,"moisture":71}	json	a few seconds ago
Data	{"temp":58,"Humid":40,"moisture":71}	json	a few seconds ago
Data	{"temp":58,"Humid":40,"moisture":71}	json	a few seconds ago
Data	{"temp":58,"Humid":40,"moisture":71}	json	a few seconds ago
Data	{"temp":58,"Humid":40,"moisture":71}	json	a few seconds ago

Node-Red Flow Diagram for Sensor:



Node-Red Flow Diagram to configure with Button:



Source Code:

```
msg.payload = { "temp": global.get("t"),
                "Humid": global.get("h"),
                "moisture": global.get("m")}

return msg;
```

For Temperature:

```
msg.payload = msg.payload.temp
global.set("t",msg.payload)
return msg;
```

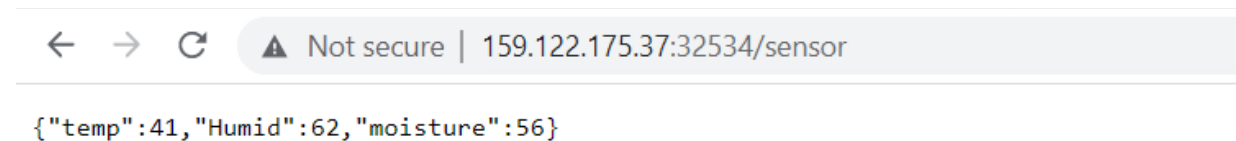
For Humidity:

```
msg.payload = msg.payload.Humid
global.set("h", msg.payload)
return msg;
```

For Moisture:

```
msg.payload = msg.payload.moisture
global.set("m", msg.payload)
return msg;
```

HTTP Request using Node-Red:



Generate the output for recent event:

Smart crop Protection

Analytics

MOTOR ON

MOTOR OFF

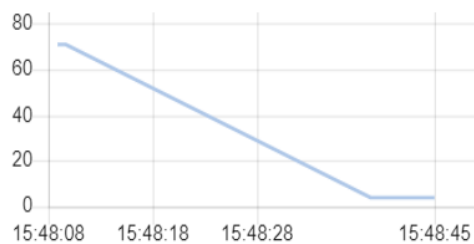
Temperature



Humidity



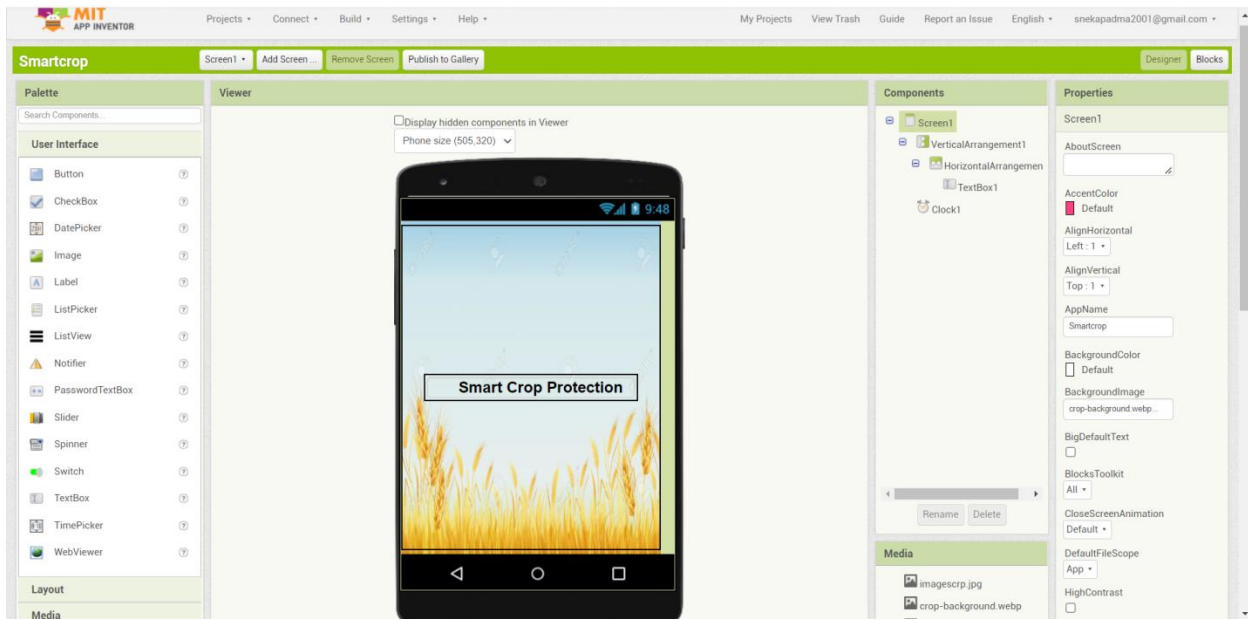
Soil Moisture



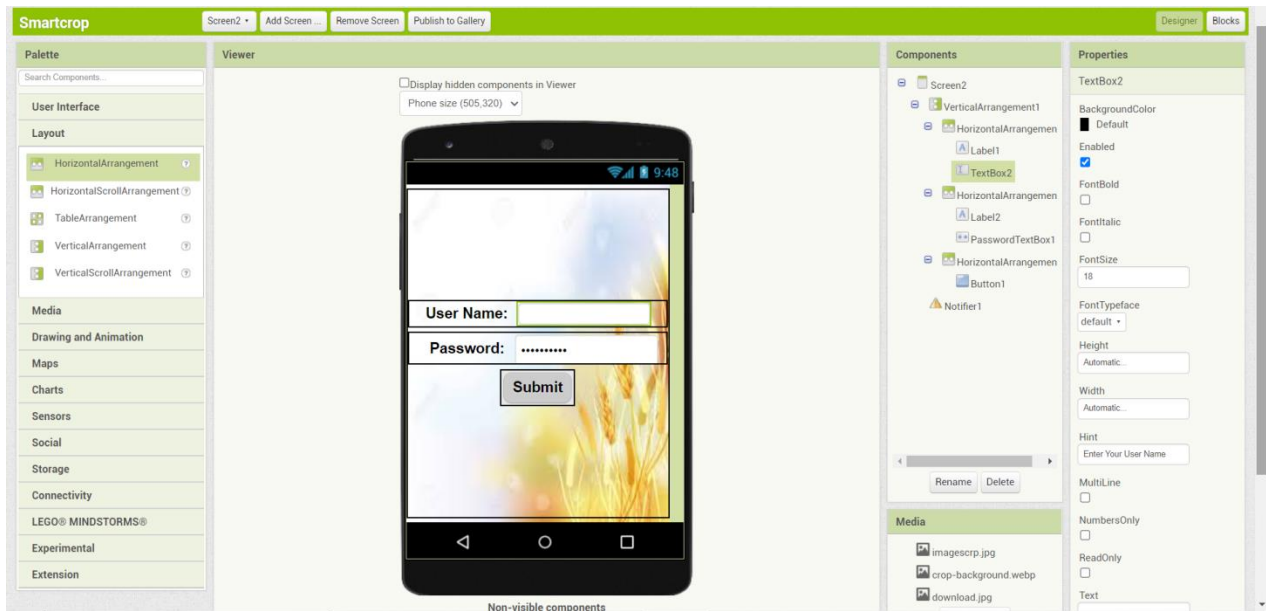
Mit Application Designer:

(FRONTEND)

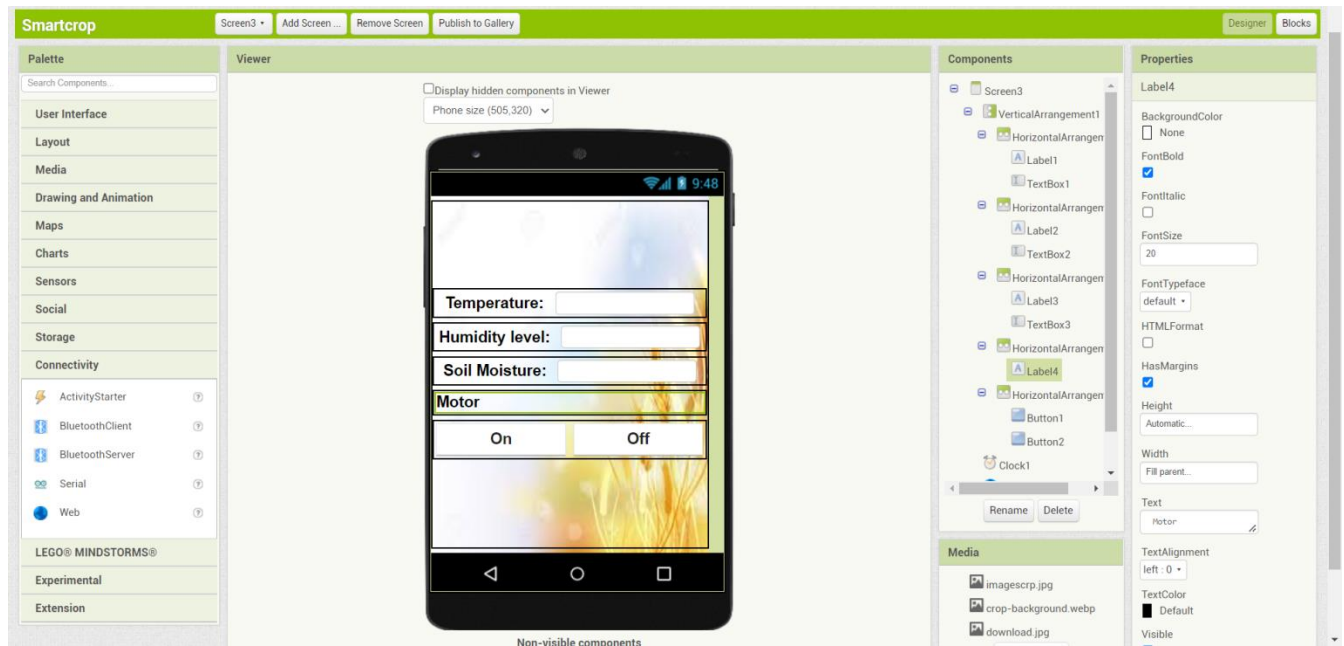
Screen-1:



Screen-2:



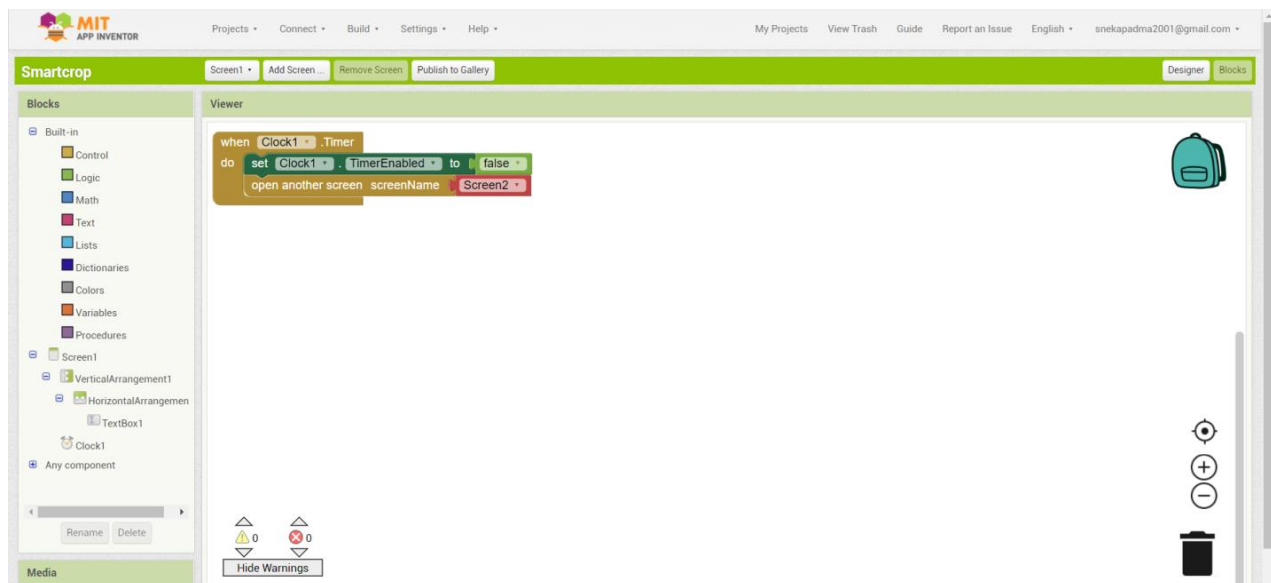
Screen-3:



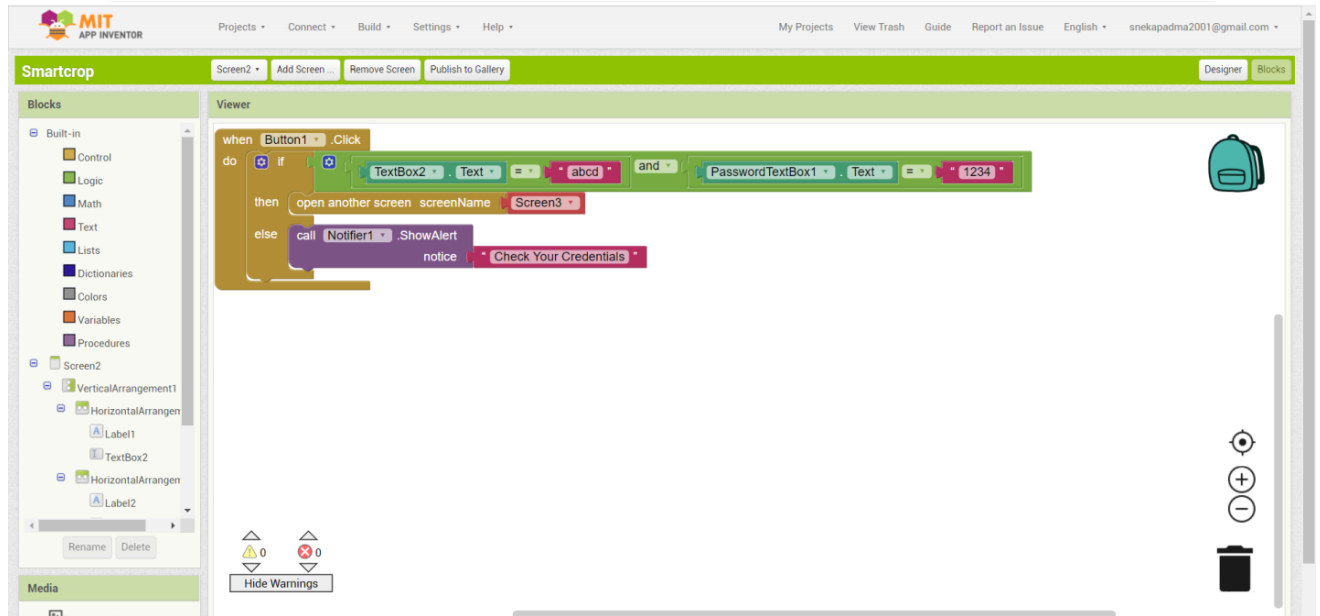
Customize the APP interface to display the values:

Blocks(BACKEND)

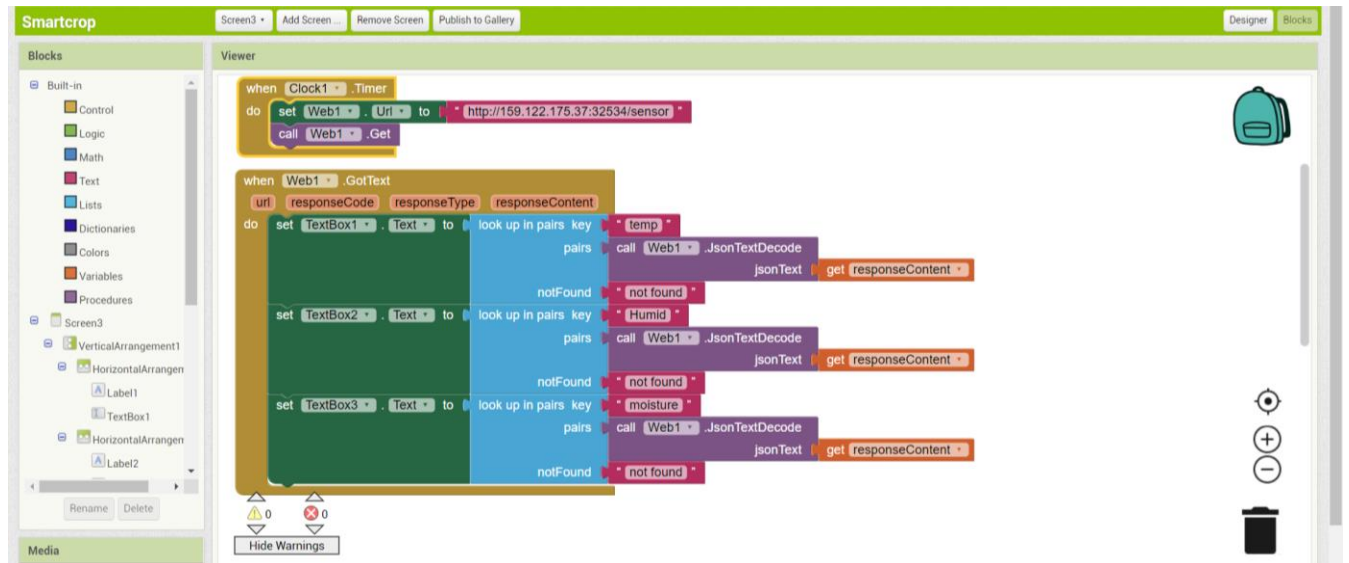
Screen-1:



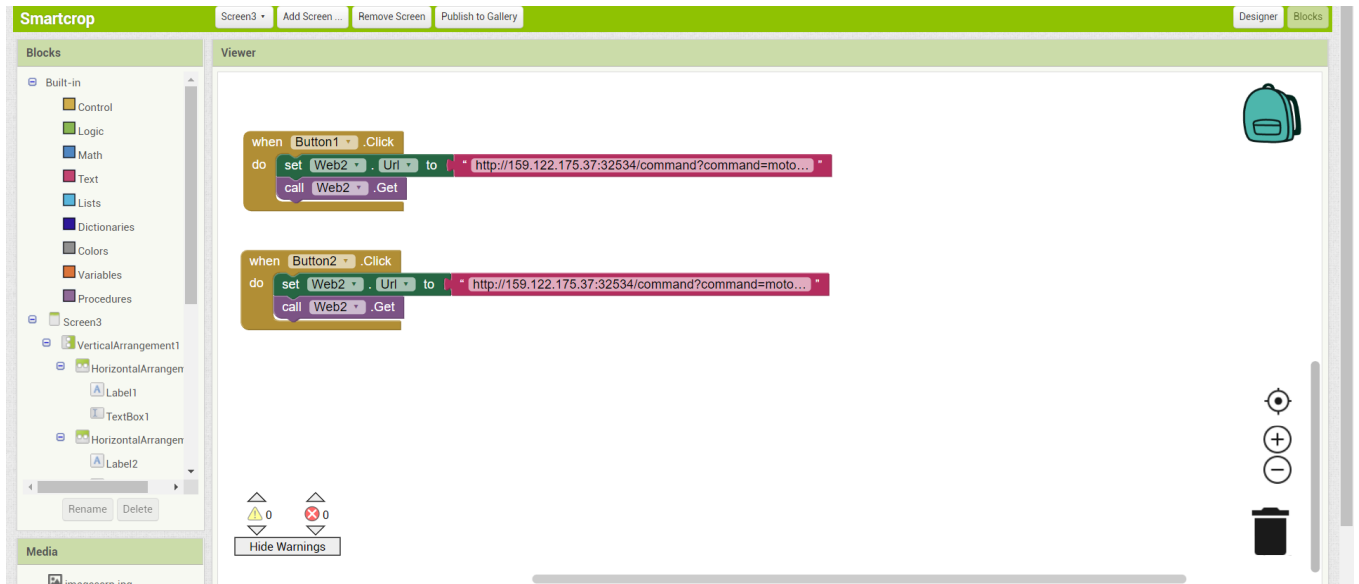
Screen-2:



Screen-3 :

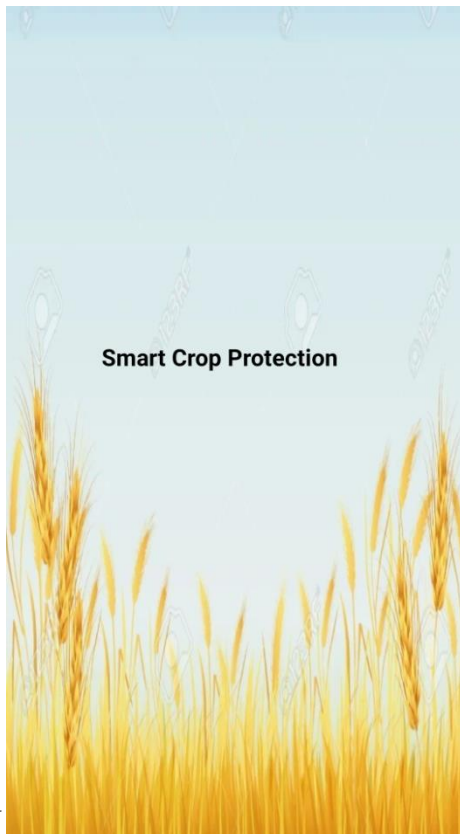


Screen-3 Blocks with Button:



MIT App Inventor Output-Mobile Phone:

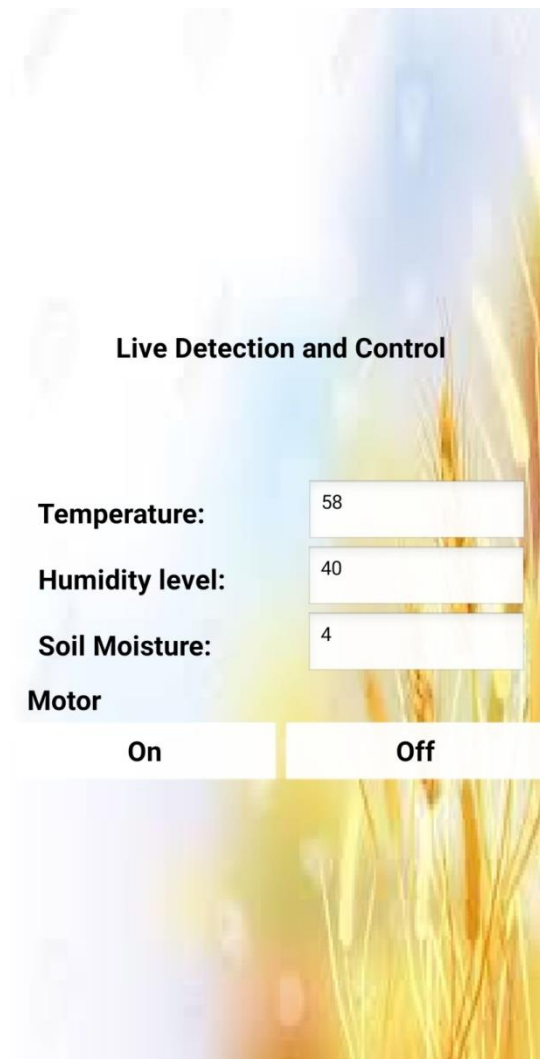
Screen-1:



Screen-2:



Screen-3:



Live Detection and Control

Temperature:

Humidity level:

Soil Moisture:

Motor