

2023

ASSIGNMENT 3

Switch on and off LED from Node Red



WOKWI FILE:

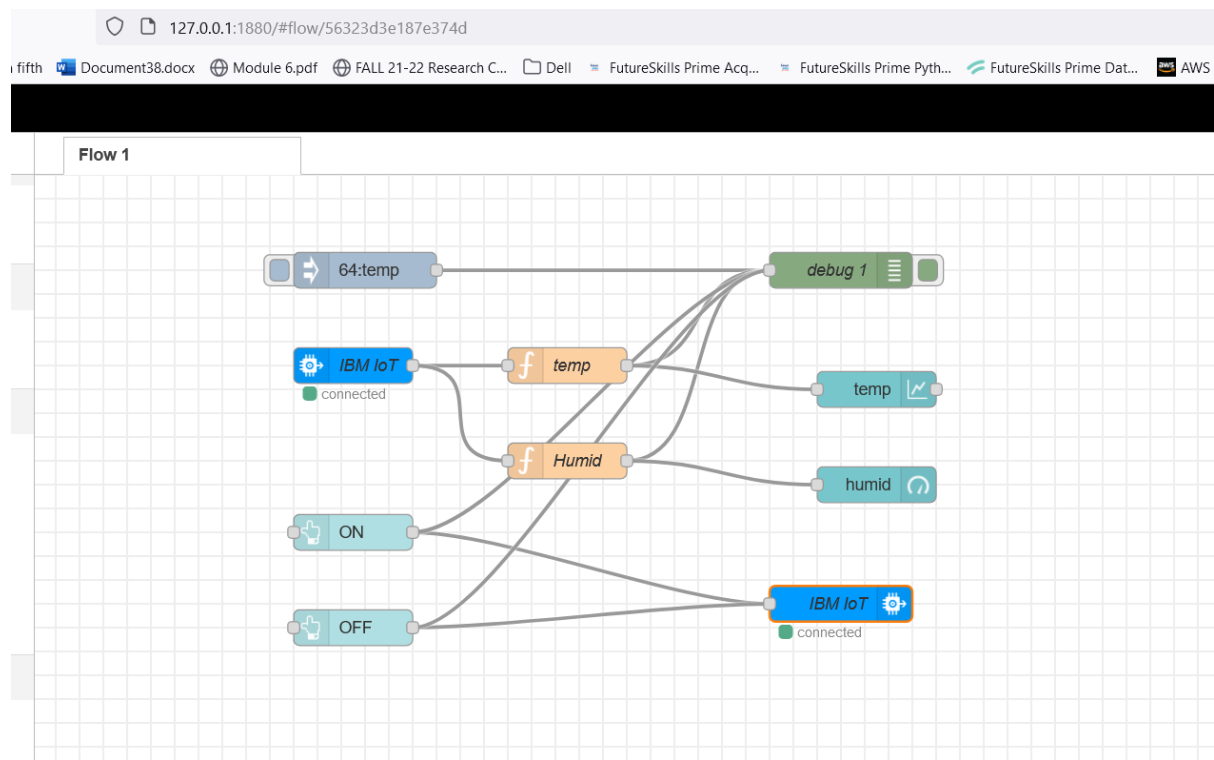
The screenshot shows the Wokwi IDE interface. On the left, the sketch.ino file is open, displaying the following code:

```
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 2
7
8 DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin and type of dht connect
9
10 void callback(char* topic, byte* payload, unsigned int payloadlength);
11
12 //-----credentials of IBM Accounts-----
13
14 #define ORG "qab2n1" //IBM ORGANIZATION ID
15 #define DEVICE_TYPE "Shreya" //Device type mentioned in ibm watson IOT Platform
16 #define DEVICE_ID "1912" //Device ID mentioned in ibm watson IOT Platform
17 #define TOKEN "1igkjlgnde" //Token
18 String data;
19 float h, t;
20
21 //----- Customise the above values -----
22 char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Server Name
23 char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and type of event perform
24 char subscribTopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT command type AND
25 char authMethod[] = "use-token-auth"; // authentication method
26 char token[] = TOKEN;
27 char clientid[] = "dt:" ORG ":" DEVICE_TYPE ":" DEVICE_ID; //client id
28
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 ESP32 board connected to a DHT22 sensor. The output log displays the following data:

```
Humid:50.00
Sending payload: {"temp":55.80,"Humid":50.00}
Publish ok
temp:55.80
Humid:50.00
Sending payload: {"temp":55.80,"Humid":50.00}
Publish ok
```

NODE-RED Configurations:



1. ibmiot in node

The screenshot shows the 'Edit ibmiot in node' configuration window. At the top, there are buttons for 'Delete', 'Cancel', and 'Done'. Below this is a 'Properties' tab with a settings icon, a document icon, and a preview icon. The properties are listed on the left with corresponding input fields on the right:

- Authentication: API Key (dropdown)
- API Key: IBMIotapi (text field with a search icon)
- Input Type: Device Event (dropdown)
- Device Type: ☒ All or + (text field)
- Device Id: ☐ All or device id e.g. ab12cd231a21 (text field)
- Event: ☒ All or + (text field)
- Format: ☐ All or json (text field)
- QoS: 0 (dropdown)
- Name: IBM IoT (text field)
- Service: registered (text field)

At the bottom, there is a yellow highlighted note: 'Use the Input Type property to configure this node to receive Events'. Below the note is an 'Enabled' checkbox which is currently unchecked.

2. function node(temp)

The screenshot shows the 'Edit function node' configuration window. At the top, there are buttons for 'Delete', 'Cancel', and 'Done'. Below this is a 'Properties' tab with a settings icon, a document icon, and a preview icon. The properties are listed on the left with corresponding input fields on the right:

- Name: temp (text field with a search icon)

Below the properties, there are four tabs: 'Setup', 'On Start', 'On Message', and 'On Stop'. The 'On Message' tab is selected. The code editor shows the following code:

```
1 msg.payload=msg.payload.temp
2 return msg;
```

At the bottom, there is an 'Enabled' checkbox which is currently unchecked.

3.Buttons(ON)

DeleteCancelDone

Properties

Group

[Home] SmartSwitchBoard

Size

auto

Icon

optional icon

Label

ON

Tooltip

optional tooltip

Color

optional text/icon color

Background

optional background color

When clicked, send:

Payload

lighton

Topic

msg. topic

If msg arrives on input, emulate a button click:

Enabled

4.Ibmiot out node

DeleteCancelDone

Properties

Authentication

API Key

API Key

IBMIotapi

Output Type

Device Command

Device Type

Shreya

Device Id

1912

Command Type

command

Format

String

Data

Data

QoS

0

Name

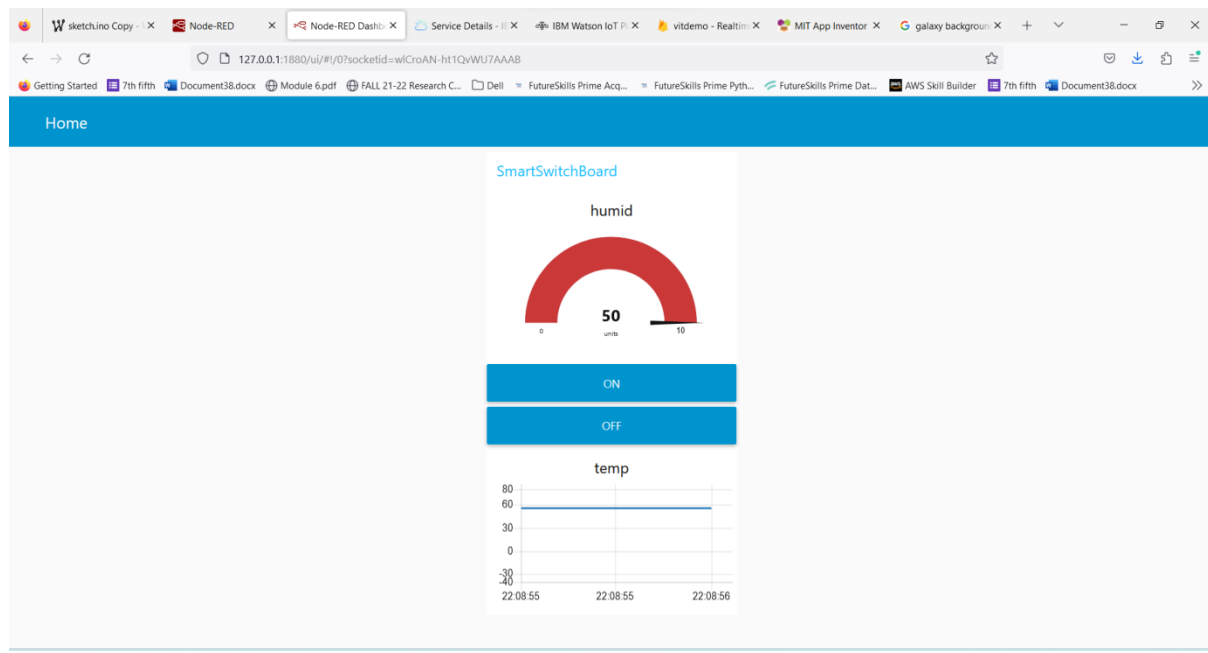
IBM IoT

Service

registered

Enabled

CHART PAGE WITH BUTTONS:



PROCESS:

Run the Wokwi file and then press the on or off button on the chart page and check on wokwi page for output.

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 "qab2n1" //IBM ORGANITION ID
#define DEVICE_TYPE "Shreya" //Device type mentioned in ibm watson IOT Platform
```

```

#define DEVICE_ID "1912"//Device ID mentioned in ibm watson IOT Platform
#define TOKEN "myp@ssword" //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="";
}

```


OUTPUT:

No buttons pressed

The screenshot shows the Wokwi IDE interface. On the left, the sketch code is displayed, which includes libraries for WiFi, MQTT, and DHT. The code defines an ESP32 board and a DHT22 sensor connected to pins 15 and 2. It sets up an MQTT client to connect to an IBM Watson IoT Platform. The simulation window on the right shows a visual representation of the ESP32 and DHT22 sensor. The console output indicates that the sensor has read a humidity of 50.00 and the payload being sent is {"temp":55.80,"Humid":50.00}.

ON button pressed:

As we can see in the console, it shows :

callback invoked for topic: iot-2/cmd/command/fmt/String

data: lighton

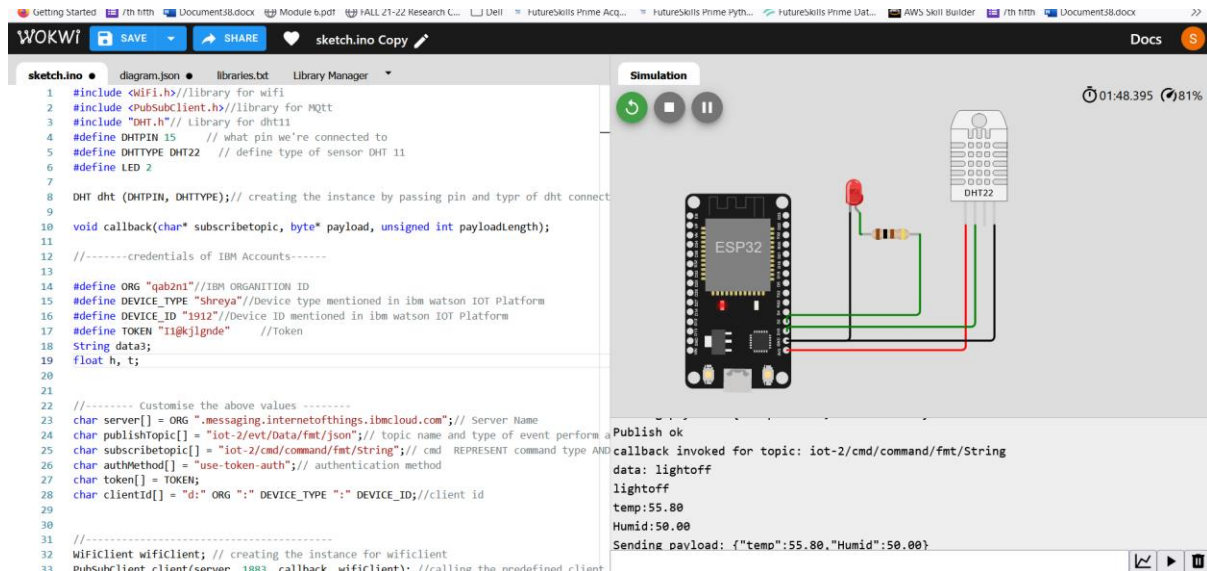
This screenshot shows the same Wokwi IDE interface as the previous one, but with a button press event triggered. The console output now shows the callback being invoked for the topic iot-2/cmd/command/fmt/String with the data 'lighton'. The simulation window shows the same ESP32 and DHT22 sensor setup, but the console output indicates that the callback has been triggered by the button press event.

OFF button pressed:

As we can see in the console, it shows :

callback invoked for topic: iot-2/cmd/command/fmt/String

data: lightoff



[Link to project:](https://wokwi.com/projects/365704565335350273)

<https://wokwi.com/projects/365704565335350273>