

Project Development Phase Sprint IV

Date	19 November 2022
Team ID	PNT2022TMID35906
Project Name	Signs with Smart Connectivity for better road safety

Wokwi Simulation: <https://wokwi.com/projects/347494122536305235>

WOKWI

SAVE SHARE final_iotino Docs SIGN IN

sketch.ino diagram.json libraries.txt Library Manager

```
1 #include <WiFi.h> // library for wifi
2 #include <PubSubClient.h> // library for MQTT
3 #include "DHT.h" // library for dht11
4 #define DHTPIN 5 // what pin we're connected to
5 #define DHTTYPE DHT22 // define type of sensor DHT 11
6
7 DHT dht (DHTPIN, DHTTYPE); // creating the instance by passing pin ar
8
9 void callback(char* subscribetopic, byte* payload, unsigned int payl
10
11 //-----credentials of IBM Accounts-----
12
13 #define ORG "psh4py" // IBM ORGANITION ID
14 #define DEVICE_TYPE "alert-device" // Device type mentioned in ibm wat
15 #define DEVICE_ID "4571" // Device ID mentioned in ibm watson IOT Plat
16 #define TOKEN "12345678" // Token
17 String data3;
18 float h, t;
19
20
21 //----- Customise the above values -----
22 char server[] = ORG ".messaging.internetofthings.ibmcloud.com"; // Se
23 char publishTopic[] = "iot-2/evt/Data/fmt/json"; // topic name and ty
24 char subscribetopic[] = "iot-2/cmd/command/fmt/String"; // cmd REPRESENT
25 char authMethod[] = "use-token-auth"; // authentication method
26 char token[] = TOKEN;
27 char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID; // client id
28
29
30 //-----
31 WiFiClient wificlient; // creating the instance for wificlient
32 PubSubClient client(server, 1883, callback, wificlient); // calling
33
34
```

Simulation

humidity:63.00
Sending payload: {"temp":12.30,"humidity":63.00,"North":0,"South":0,"East":0,"West":1}
Publish ok
temp:12.30
humidity:63.00
Sending payload: {"temp":12.30,"humidity":63.00,"North":0,"South":0,"East":0,"West":1}
Publish ok

Wokwi Simulation for Traffic Light Control

<https://wokwi.com/projects/348774600326251090>

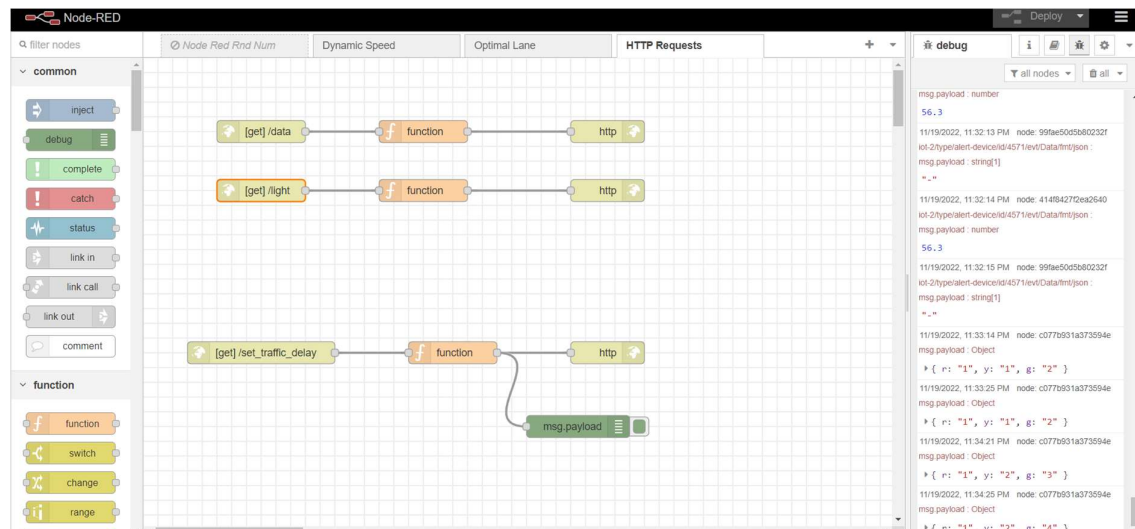
The screenshot displays the Wokwi simulation interface. On the left, the code editor shows the following C++ code for an ESP32:

```
26 HTTPClient http;
27
28 if(WiFi.status() == WL_CONNECTED){
29   // Your Domain name with URL path or IP address with path
30   http.begin(serverName.c_str());
31   int httpResponseCode = http.GET();
32   if (httpResponseCode > 0) {
33     String payload = http.getString();
34     Serial.println(payload);
35     deserializeJson(doc, payload);
36     dr = doc["r"]; dy = doc["y"]; dg = doc["g"];
37     dr = dr * 1000;
38     dy = dy * 1000;
39     dg = dg * 1000;
40   }
41 }
42 Serial.println(dr);
43 Serial.println(dy);
44 Serial.println(dg);
45 digitalWrite(RED, HIGH);
46 delay(dr);
47 digitalWrite(RED, LOW);
48
49 digitalWrite(YELLOW, HIGH);
50 delay(dy);
51 digitalWrite(YELLOW, LOW);
52
53 digitalWrite(GREEN, HIGH);
54 delay(dg);
55 digitalWrite(GREEN, LOW);
56 }
57
58 void wificonnect() //function definition for wificonnect
59 {
60   Serial.println();
```

On the right, the simulation window shows an ESP32 board connected to three LEDs: Red, Yellow, and Green. The Green LED is currently lit. Below the simulation, a console window displays the following output:

```
1000
2000
4000
{"r": "1", "y": "2", "g": "4"}
1000
2000
4000
```

Node Red



MIT App Inventor

