```
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>

#define readyPin 2
// #define WDT_TIMEOUT 20000

// WiFi credentials
const char *ssid = "Home Base";
const char *password = "leb87/11";

// MQTT Broker
const char *mqtt_server = "emqx.home.mwaleedh.com.pk";
const int mqtt_port = 8883; // MQTT port
const char *mqtt_user = "waleed";
const char *mqtt_password = "Hrmzz@990066";
const char *mqtt_client_id = "esp01_waleed";
bool switchState = false; // Initial state
float temperature = 0.0;  // Initial temperature value

WiFiClientSecure espClient;
PubSubClient client(espClient);

void reconnect() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect(mqtt_client_id, mqtt_user, mqtt_password)) {
      Serial.println("MQTT connected");
      digitalWrite(readyPin, HIGH);
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retry in 5 seconds");
      delay(5000);
    }
  }
}

void callback(char *topic, byte *payload, unsigned int length) {
  Serial.print("Message received on topic: ");
  Serial.println(topic);

  // Parse JSON payload
  JsonDocument doc;
  DeserializationError error = deserializeJson(doc, payload, length);
  if (error) {
    Serial.print("deserializeJson() failed: ");
```

```
    Serial.println(error.c_str());
    return;
  }
  // digitalWrite(LED, LOW);

  // Extract values from JSON and update variables
  // switchState = doc["switch"];
  // temperature = doc["temperature"];

  // Serial.print("Switch state: ");
  // Serial.println(switchState);
  // Serial.print("Temperature: ");
  // Serial.println(temperature);
  delay(1000);
}
unsigned long previousMillis = 0;

void nonBlockingDelay(unsigned long interval) {
  static unsigned long previousMillis = 0;
  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    // Perform action here...
  }
}

void deviceDiscoveryHA() {
  char topic[128];
  char buffer1[512];
  char buffer2[512];
  char buffer3[512];
  char buffer4[512];
  char uid[128];
  JsonDocument doc;
  doc.clear();
  // creating topic here
  strcpy(topic, "homeassistant/binary_sensor/");
  strcat(topic, mqtt_client_id);
  strcat(topic, "_BS/config");

  // creating payload for Window Sensor
  strcpy(uid, mqtt_client_id);
  strcat(uid, "_BS");
  doc["name"] = "Window Sensor";
  doc["obj_id"] = "mqtt_window_sensor";
  doc["uniq_id"] = uid;
  doc["stat_t"] = "esp01_waleed/sensors/window_sensor";
```

```
  doc["value_template"] = "{{value_json.state}}";
  doc["payload_on"] = "close";
  doc["payload_off"] = "open";
  doc["payload_available"] = "available";
  doc["not_payload_available"] = "not_available";

  JsonObject device = doc.createNestedObject("device");
  device["ids"] = mqtt_client_id;
  device["name"] = "Sensing Device";
  device["mf"] = "Waleed";
  device["mdl"] = "ESP01";
  device["sw"] = "0.2";
  device["hw"] = "1.0";
  // device["cu"] = "http://192.168.1.226/config";  //web interface for
device,
  // with discovery toggle
  serializeJson(doc, buffer1);
  // Publish discovery topic and payload (with retained flag)
  client.publish(topic, buffer1, true);

  // Creating topic for light sensor
  doc.clear();
  // creating topic here
  strcpy(topic, "homeassistant/sensor/");
  strcat(topic, mqtt_client_id);
  strcat(topic, "_LS/config");

  // creating payload for Light Sensor
  strcpy(uid, mqtt_client_id);
  strcat(uid, "_LS");
  doc["name"] = "Light Sensor";
  doc["obj_id"] = "mqtt_light_sensor";
  doc["dev_cla"] = "illuminance";
  doc["uniq_id"] = uid;
  doc["stat_t"] = "esp01_waleed/sensors/lightlevel";
  doc["unit_of_meas"] = "lx";
  doc["value_template"] = "{{value_json.lux}}";
  doc["not_payload_available"] = "not_available";

  JsonObject deviceL = doc.createNestedObject("device");
  deviceL["ids"] = mqtt_client_id;
  deviceL["name"] = "Sensing Device";
  serializeJson(doc, buffer2);
  // Publish discovery topic and payload (with retained flag)
  client.publish(topic, buffer2, true);

  // creating topic for humidity sensor
  doc.clear();
```

```
  // creating topic here
  strcpy(topic, "homeassistant/sensor/");
  strcat(topic, mqtt_client_id);
  strcat(topic, "_HM/config");

  // creating payload for humidity Sensor
  strcpy(uid, mqtt_client_id);
  strcat(uid, "_HM");
  doc["name"] = "Humidity";
  doc["obj_id"] = "mqtt_RH_sensor";
  doc["dev_cla"] = "humidity";
  doc["uniq_id"] = uid;
  doc["stat_t"] = "esp01_waleed/sensors/TH_sensor";
  doc["unit_of_meas"] = "%";
  doc["value_template"] = "{{value_json.humidity}}";

  JsonObject deviceH = doc.createNestedObject("device");
  deviceH["ids"] = mqtt_client_id;
  deviceH["name"] = "Sensing Device";
  serializeJson(doc, buffer3);
  // Publish discovery topic and payload (with retained flag)
  client.publish(topic, buffer3, true);

  // creating topic for temprature sensor
  doc.clear();
  // creating topic here
  strcpy(topic, "homeassistant/sensor/");
  strcat(topic, mqtt_client_id);
  strcat(topic, "_TS/config");

  // creating payload for temprature Sensor
  strcpy(uid, mqtt_client_id);
  strcat(uid, "_TS");
  doc["name"] = "Temprature";
  doc["obj_id"] = "mqtt_temprature_sensor";
  doc["dev_cla"] = "temperature";
  doc["uniq_id"] = uid;
  doc["stat_t"] = "esp01_waleed/sensors/TH_sensor";
  doc["unit_of_meas"] = "°C";
  doc["value_template"] = "{{value_json.temprature}}";

  JsonObject deviceT = doc.createNestedObject("device");
  deviceT["ids"] = mqtt_client_id;
  deviceT["name"] = "Sensing Device";
  serializeJson(doc, buffer4);
  // Publish discovery topic and payload (with retained flag)
  client.publish(topic, buffer4, true);
}
```

```
void setup() {
  Serial.begin(115200);
  Serial.println("Turning On...");
  delay(5000);

  WiFi.begin(ssid, password);
  Serial.println("Wifi Function called");
  while (WiFi.status() != WL_CONNECTED) {
    nonBlockingDelay(50000);
    Serial.println("Connecting to WiFi...");
    // delay(5000);
  }
  Serial.println("WiFi connected");

  // if (root_ca != NULL) {
  //   espClient.setCACert(root_ca);
  // } else {
  espClient.setInsecure();
  //}
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  client.setBufferSize(512); // increasing buffer size

  while (!client.connected()) {
    Serial.println("Connecting to MQTT...");
    if (client.connect(mqtt_client_id, mqtt_user, mqtt_password)) {
      Serial.println("MQTT connected");
      deviceDiscoveryHA();
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" Retry in 5 seconds");
      delay(5000);
    }
  }
  // send initial config message here, intial subscribe here
  // client.subscribe("topic_name");
  // Serial.println("Subscribed to topic");
  // client.publish(topic.c_str(), message.c_str());

  // action config here
  // pinMode(LED, OUTPUT);
}

void loop() {
  // ESP.wdtFeed();
  if (!client.connected()) {
```

```
    reconnect();
  }
  client.loop();
  // digitalWrite(LED, HIGH);
  //  Read switch state and temperature from serial
  if (Serial.available() > 0) {
    String input = Serial.readStringUntil('\n');
    if (input.startsWith("publish:")) {
      int separatorIndex = input.indexOf('|');
      if (separatorIndex != -1) {
        String topic = input.substring(8, separatorIndex);
        String payload = input.substring(separatorIndex + 1);
        JsonDocument doc;
        DeserializationError error = deserializeJson(doc, payload);
        if (!error) {
          switchState = doc["switch"];
          temperature = doc["temperature"];
          client.publish(topic.c_str(), payload.c_str());
          Serial.println("Published message:");
          Serial.println(payload);
        }
        //   // char buffer[256];
        //   // size_t n = serializeJson(doc, buffer);
      }
    } else if (input.startsWith("subscribe:")) {
      String topic = input.substring(10);
      client.subscribe(topic.c_str());
      Serial.print("Subscribed to topic: ");
      Serial.println(topic);
    } else if (input.startsWith("unsubscribe:")) {
      String topic = input.substring(12);
      client.unsubscribe(topic.c_str());
      Serial.print("Unsubscribed from topic: ");
      Serial.println(topic);
    }
  }
}
```