LAMPIRAN

Source Code :

1. Js.php

```
          <script
src="https://cdn.jsdelivr.net/npm/moment@2.24.0/min/moment.min.js"></script>
                    <script src="https://cdn.jsdelivr.net/npm/chart.js@2.8.0"></script>
             <script src="https://cdn.jsdelivr.net/npm/chartjs-plugin-
streaming@1.8.0"></script>
             <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-
switch/3.3.4/js/bootstrap-switch.js" data-turbolinks-track="true"></script>
          <script>
             $("input[data-bootstrap-switch]").each(function(){
               $(this).bootstrapSwitch('state', $(this).prop('checked'));
             })
          </script>

             <script type="text/javascript">
                    function showTime() {
                       var a_p = "";
                       var today = new Date();
                       var curr_hour = today.getHours();
                       var curr_minute = today.getMinutes();
                       var curr_second = today.getSeconds();
                       if (curr_hour < 12) {
                          a_p = "AM";
                       } else {
                          a_p = "PM";
                       }
                       if (curr_hour == 0) {
                          curr_hour = 12;
                       }
                       if (curr_hour > 12) {
                          curr_hour = curr_hour - 12;
                       }
                       curr_hour = checkTime(curr_hour);
                       curr_minute = checkTime(curr_minute);
                       curr_second = checkTime(curr_second);
                       document.getElementById('time').innerHTML=curr_hour + ":" +
curr_minute + ":" + curr_second + " " + a_p;
                    }

                    function checkTime(i) {
                       if (i < 10) {
                          i = "0" + i;
                       }
                       return i;
                    }
                    setInterval(showTime, 500);
            </script>

            <script src="https://cdnjs.cloudflare.com/ajax/libs/paho-
mqtt/1.0.2/mqttws31.min.js" type="text/javascript"></script>
            <script type="text/javascript">
               var MQTTbroker = '44.195.141.13';
                     // var dataTopics = new Array();
               var messagePayloadTemperature = 0;
               var messagePayloadHumidity = 0;
               var messagePayloadMq = 0;

               var client = new Paho.MQTT.Client(MQTTbroker, 9095, "myclientid_" +
parseInt(Math.random() * 100, 10));

               //mqtt connecton options including the mqtt broker subscriptions
               client.connect ({
                  onSuccess: function () {
                     console.log("mqtt connected");
                     // Connection succeeded; subscribe to our topics
                     // client.subscribe(MQTTsubTopic, {qos: 1});
                     client.subscribe("gas/iot/temperature");
                     client.subscribe("gas/iot/humidity");
                     client.subscribe("gas/iot/mq");
                     client.subscribe("gas/iot/mac");
                     client.subscribe("nodemcu/kipas");
                     client.subscribe("nodemcu/buzzer");
                     client.subscribe("nodemcu/manual");
                     client.subscribe("nodemcu/device");
                     //topic mac address

                     client.onMessageArrived = onMessageArrived;
                     client.onConnectionLost = onConnectionLost;
                  },

                  onFailure: function (message) {
                     console.log("Connection failed, ERROR: " + message.errorMessage);
                     //window.setTimeout(location.reload(),20000); //wait 20seconds before
trying to connect again.
                  }
               });

               //can be used to reconnect on connection lost
               function onConnectionLost(responseObject) {
                  console.log("connection lost: " + responseObject.errorMessage);
                  //window.setTimeout(location.reload(),20000); //wait 20seconds before trying
to connect again.
               };

               //what is done when a message arrives from the broker
```

```
          function onMessageArrived(message) {
            console.log(message.destinationName, '',message.payloadString);

            //check if it is a new topic, if not add it to the array

            if(message.destinationName == "gas/iot/temperature"){
              console.log("Message Arrived : " + message.payloadString);
              // document.getElementById("temperature").innerHTML = '<span>' +
message.payloadString +' </span>';
              messagePayloadTemperature = parseInt(message.payloadString);
              // Creating a cookie after the document is ready
              $(document).ready(function () {
                createCookie("suhu", messagePayloadTemperature, "10");
              });

              // Function to create the cookie
              function createCookie(name, value, days) {
                var expires;

                if (days) {
                  var date = new Date();
                  date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
                  expires = "; expires=" + date.toGMTString();
                }
                else {
                  expires = "";
                }

                document.cookie = escape(name) + "=" +
                  escape(value) + expires + "; path=/";
              }
              console.log("Temperature: " + messagePayloadTemperature);

            } if(message.destinationName == "gas/iot/humidity"){
              console.log("Message Arrived : " + message.payloadString);
              // document.getElementById("humidity").innerHTML = '<span>' +
message.payloadString +' </span>';
              messagePayloadHumidity = parseInt(message.payloadString);
              // Creating a cookie after the document is ready
              $(document).ready(function () {
                createCookie("hum", messagePayloadHumidity, "10");
              });

              // Function to create the cookie
              function createCookie(name, value, days) {
                var expires;

                if (days) {
                  var date = new Date();
                  date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
```

```
                        expires = "; expires=" + date.toGMTString();
                    }
                    else {
                        expires = "";
                    }

                    document.cookie = escape(name) + "=" +
                        escape(value) + expires + "; path=/";
                }
                console.log("Humidity: " + messagePayloadHumidity);

            } if(message.destinationName == "gas/iot/mq"){
                console.log("Message Arrived : " + message.payloadString);
                // document.getElementById("mq2").innerHTML = '<span>' +
message.payloadString +' </span>';
                messagePayloadMq = parseInt(message.payloadString);
                // Creating a cookie after the document is ready
                $(document).ready(function () {
                    createCookie("gas", messagePayloadMq, "10");
                });

                // Function to create the cookie
                function createCookie(name, value, days) {
                    var expires;

                    if (days) {
                        var date = new Date();
                        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
                        expires = "; expires=" + date.toGMTString();
                    }
                    else {
                        expires = "";
                    }

                    document.cookie = escape(name) + "=" +
                        escape(value) + expires + "; path=/";
                }
                console.log("mq2: " + messagePayloadMq);
            }
            if(message.destinationName== "gas/iot/mac"){
                console.log("Message Arrived: " + message.payloadString);
                // document.getElementById("coba").innerHTML  = '<span>'
+message.payloadString +' </span>';
                messagePayloadMac = message.PayloadString;
                if(message.payloadString == <?php echo
json_encode($_SESSION["id_perangkat"]);?>){
                    // document.getElementById("coba").innerHTML  = '<span>'
+message.payloadString +' </span>';
                    document.getElementById("temperature").innerHTML  = '<span>'
+messagePayloadTemperature +' </span>';
```

```
                    document.getElementById("humidity").innerHTML  = '<span>'
+messagePayloadHumidity +' </span>';
                    document.getElementById("mq").innerHTML  = '<span>'
+messagePayloadMq +' </span>';
                }
            }
        };

        function publishToMQTT(message) {
            message = new Paho.MQTT.Message(message ? "1" : "0");
            message.destinationName = "nodemcu/manual";
            client.send(message);
        }

        function publishToMQTT_Kipas(message) {
            message = new Paho.MQTT.Message(message);
            message.destinationName = "nodemcu/kipas";
            client.send(message);
        }

        function publishToMQTT_Buzzer(message) {
            message = new Paho.MQTT.Message(message);
            message.destinationName = "nodemcu/buzzer";
            client.send(message);
        }
        function publishToMQTT_de() {
            var device = <?php echo json_encode($_SESSION["id_perangkat"]);?>;
            message = new Paho.MQTT.Message(device);
            message.destinationName = "nodemcu/device";
            client.send(message);
        }
        $(document).ready(function () {
            $("#manualBtn").bootstrapSwitch();

            $('#manualBtn').on('switchChange.bootstrapSwitch', function (event, state) {
                publishToMQTT(state);
                publishToMQTT_de();
            });
        });

        $(document).ready(function () {
            setInterval(function() {
                $("#kipas").load('sugeno.php');
            }, 10000);
            setInterval(function() {
                publishToMQTT_Kipas(document.getElementById("kipas").innerHTML);
            }, 10000);

            setInterval(function() {
                $("#buzzer").load('sugeno2.php');
            }, 10000);
            setInterval(function() {

publishToMQTT_Buzzer(document.getElementById("buzzer").innerHTML);
            }, 10000);
        });

        function refreshTemperature(chart){
            chart.config.data.datasets.forEach(function (dataset){
                dataset.data.push({
                    x: Date.now(),
                    y: messagePayloadTemperature
                });
            });
        }

        function onrefreshHum(chart){
            chart.config.data.datasets.forEach(function (dataset){
                dataset.data.push({
                    x: Date.now(),
                    y: messagePayloadHumidity
                })
            });
        }

        function onrefreshMq(chart){
            chart.config.data.datasets.forEach(function (dataset){
                dataset.data.push({
                    x: Date.now(),
                    y: messagePayloadMq
                })
            });
        }
        var chartColors = {
            red: 'rgb(255, 99, 132)',
            orange: 'rgb(255, 159, 64)',
            yellow: 'rgb(255, 205, 86)',
            green: 'rgb(75, 192, 192)',
            blue: 'rgb(54, 162, 235)',
            purple: 'rgb(153, 102, 255)',
            grey: 'rgb(201, 203, 207)'
        };
        var color = Chart.helpers.color;
        var configTemperature = {
            type: 'line',
            data: {
                datasets: [{
                    label: 'Temperature',
                                    backgroundColor:
color(chartColors.red).alpha(0.5).rgbString(),
                    borderColor: chartColors.yellow,
                    fill: false,
                    // lineTension: 0,
                    // borderDash: [8, 4],
                    data: []
                }]
            },
            options: {
                title: {
                    display: true,
                    // text: "Temperature"
                },
                scales: {
                    xAxes: [{
                        type: 'realtime',
                        realtime: {
                            duration: 20000,
                            refresh: 2000,
                            delay: 3000,
                            onRefresh: refreshTemperature
                        }
                    }],
                    yAxis: [{
                        title: {
                            display: true,
                            text: 'Value'
                        }
                    }]
                },
                tooltips: {
                    mode: 'nearest',
                    intersect: false
                },
                hover: {
                    mode: 'nearest',
                    intersect: false
                }
            }
        };

        var configHumidity = {
            type: 'line',
            data: {
                datasets: [{
                    label: 'Humidity',
                                    backgroundColor:
color(chartColors.grey).alpha(0.5).rgbString(),
                    borderColor: chartColors.blue,
                    fill: false,
                    // lineTension: 0,
                    // borderDash: [8, 4],
                    data: []
                }]
            },
            options: {
                title: {
                    display: true,
                    // text: "Temperature"
                },
                scales: {
                    xAxes: [{
                        type: 'realtime',
                        realtime: {
                            duration: 20000,
                            refresh: 2000,
                            delay: 3000,
                            onRefresh: onrefreshHum
                        }
                    }],
                    yAxis: [{
                        title: {
                            display: true,
                            text: 'Value'
                        }
                    }]
                },
                tooltips: {
                    mode: 'nearest',
                    intersect: false
                },
                hover: {
                    mode: 'nearest',
                    intersect: false
                }
            }
        };

        var configMq = {
            type: 'line',
            data: {
                datasets: [{
                    label: 'Gas',
                                        backgroundColor:
color(chartColors.yellow).alpha(0.5).rgbString(),
                    borderColor: chartColors.orange,
                    fill: false,
                    // lineTension: 0,
                    // borderDash: [8, 4],
                    data: []
                }]
            },
            options: {
                title: {
                    display: true,
                    // text: "Temperature"
                },
                scales: {
                    xAxes: [{
                        type: 'realtime',
                        realtime: {
                            duration: 20000,
                            refresh: 2000,
                            delay: 3000,
                            onRefresh: onrefreshMq
                        }
                    }],
                    yAxis: [{
                        title: {
                            display: true,
                            text: 'Value'
                        }
                    }]
                },
                tooltips: {
                    mode: 'nearest',
                    intersect: false
                },
                hover: {
                    mode: 'nearest',
                    intersect: false
                }
            }
        };

        window.onload = function() {
            var ctx = document.getElementById("ChartTemperature").getContext("2d");
            window.ChartTemperature = new Chart(ctx, configTemperature);
            var ctx1 = document.getElementById("ChartHumidity").getContext("2d");
            window.ChartHumidity = new Chart(ctx1, configHumidity);
            var ctx2 = document.getElementById("ChartMq").getContext("2d");
            window.ChartMq = new Chart(ctx2, configMq);
        };
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-switch/3.3.4/js/bootstrap-
switch.js" data-turbolinks-track="true"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.10.3/jquery-
ui.min.js"></script>
    <script
src="http://cdnjs.cloudflare.com/ajax/libs/summernote/0.8.2/summernote.js"></script>
```

```
	<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm/daterangepicker/daterangepicker.min.js"></script>
	<script src="https://cdnjs.cloudflare.com/ajax/libs/jQuery-
Knob/1.2.13/jquery.knob.min.js" integrity="sha512-
NhRZzPdzMOMf005Xmd4JonwPftz4Pe99mRVcFeRDcdCtfjv46zPIi/7ZKScbpHD/V0HB1
Eb+ZWigMqw94VUVaw==" crossorigin="anonymous" referrerpolicy="no-
referrer"></script>
	<script src="https://cdnjs.cloudflare.com/ajax/libs/jqvmap/1.5.1/jquery.vmap.min.js"
integrity="sha512-
Zk7h8Wpn6b9LpplWXq1qXpnzJl8gHPfZFf8+aR4aO/4bcOD5+/Si4iNu9qE38/t/j1qFKJ08K
WX34d2xmG0jrA==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
	<!-- Bootstrap Switch -->
	<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-switch/3.3.4/js/bootstrap-
switch.js" data-turbolinks-track="true"></script>
```

2. Add divice

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/paho-mqtt/1.0.2/mqttws31.min.js"
type="text/javascript"></script>
<script type="text/javascript">

const MQTTbroker = '44.195.141.13';
  var client = new Paho.MQTT.Client(MQTTbroker, 9095, "myclientid_" +
parseInt(Math.random() * 100, 10));

  //mqtt connecton options including the mqtt broker subscriptions
  client.connect({
    onSuccess: function () {
      console.log("mqtt connected");
      client.subscribe("nodemcu/connect");
      client.subscribe("nodemcu/disconnect");
      client.subscribe("nodemcu/dis");
      client.subscribe("nodemcu/mac");

      client.onMessageArrived = onMessageArrived;
      client.onConnectionLost = onConnectionLost;
    },
    onFailure: function (message) {
      console.log("Connection failed, ERROR: " + message.errorMessage);
      //window.setTimeout(location.reload(),20000); //wait 20seconds before trying to connect
again.
    }
  });

  function onConnectionLost(responseObject) {
    console.log("connection lost: " + responseObject.errorMessage);
    //window.setTimeout(location.reload(),20000); //wait 20seconds before trying to connect
again.
  };

  function onMessageArrived(message) {
    console.log(message.destinationName, '',message.payloadString);
  }

  function publishMQTT_Connect(message){
    // alert(message);
    var cond = message.toString();
    // alert(cond);
    message = new Paho.MQTT.Message(cond);
    message.destinationName = "nodemcu/connect";
    client.send(message);

  }

  function publishMQTT_Disconnect(message){
    // alert(message);
    var cond = message.toString();
    // alert(cond);
    message = new Paho.MQTT.Message(cond);
    message.destinationName = "nodemcu/disconnect";
    client.send(message);

  }

  function publishMQTT_Mac(){
    var macNumber = document.getElementById('id_perangkat').selectedOptions[0].value;
    // alert(macNumber);
    // var macDis = document.getElementById('macadd').textContent;
    // if(!empty(macNumber)){
    message = new Paho.MQTT.Message(macNumber);
    //   alert(message);
    // } if(!empty(macDis)){
    //   message = new Paho.MQTT.Message(macDis);
    // }
    message.destinationName = "nodemcu/mac";
    client.send(message);

  }

  function publishMQTT_Dis(message){
    // var macDis = document.getElementById('macadd').textContent;
    var macDis = message.toString();
    message = new Paho.MQTT.Message(macDis);
    // alert(macDis);
    message.destinationName = "nodemcu/dis";
    client.send(message);
  }

</script>
<script src="https://cdn.datatables.net/1.10.25/js/jquery.dataTables.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-switch/3.3.4/js/bootstrap-
switch.js" data-turbolinks-track="true"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.10.3/jquery-
ui.min.js"></script>
<script
src="http://cdnjs.cloudflare.com/ajax/libs/summernote/0.8.2/summernote.js"></script>
<script type="text/javascript"
src="https://cdn.jsdelivr.net/npm/daterangepicker/daterangepicker.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jQuery-Knob/1.2.13/jquery.knob.min.js"
integrity="sha512-
NhRZzPdzMOMf005Xmd4JonwPftz4Pe99mRVcFeRDcdCtfjv46zPIi/7ZKScbpHD/V0HB1
Eb+ZWigMqw94VUVaw==" crossorigin="anonymous" referrerpolicy="no-
referrer"></script>
```

3. Arduino IDE

```
#include <DHT.h>
#include <WiFi.h>
#include <PubSubClient.h>

#define DHTTYPE DHT22
uint8_t pinDHT = 4;
int Sensor_Gas = 34;
int kipas = 27;
int buzzer = 26;
int de = 100;
DHT dht(pinDHT, DHTTYPE);



//WiFi
const char* ssid = "DEWANDARU99";
const char* pass = "999999999";

//MQTT
const char* mqtt_server = "44.195.141.13"; //IP of the MQTT broker
const char* humTopic = "gas/humidity";
const char* temTopic = "gas/temperature";
const char* gasTopic = "gas/mq";
const char* macTopic = "gas/mac";
const char* macPub = "mac";
const char* hTopic = "gas/iot/humidity";
const char* tTopic = "gas/iot/temperature";
const char* macTpc = "gas/iot/mac";
const char* gTopic = "gas/iot/mq";
const char* WebTopic = "nodemcu/#";
const char* mqtt_username = "aditya";
const char* mqtt_password = "12345678";
const char* clientID = "client_gas";



//Initialise the WiFi and MQTT Client objects
WiFiClient wifiClient;

//1883 is the listener port of the Broker
PubSubClient client(wifiClient);
long lastMsg = 0;
int value = 0;
//variabel Mac Address
String macAdd = WiFi.macAddress();
String mcAddress;
String dis;
String conn;
String hasilmac;
String manual;
String konek;
int hasil;
int hasil2;

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.print(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi Connected");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT Connection...");
    //Attempt to connect
    if (client.connect(clientID)) {
      Serial.println("connected");
      //Once connected, publish an announcement

      if (client.publish(macPub, String(macAdd).c_str())) {
        Serial.println("Mac Address sent!");
      }
      // Again, client.publish will return a boolean value depending on whether it succeded or
not.
      // If the message failed to send, we will try again, as the connection may have broken.
      else {
        client.connect(clientID);
        delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
        client.publish(macPub, String(macAdd).c_str());
      }

      client.subscribe(WebTopic);
    } else {
      Serial.print("Failed, rc= ");
      Serial.println(client.state());
      Serial.println("Try Again in 2 seconds");
      delay(2000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String top = (String)topic;
  String message;
  String mess;
  String maa;
  String disc;
  int sta;
  int s;
  int st;
  Serial.print("Message arrived in topic: ");
  if (top.equals("nodemcu/manual")) {
    manual = "";
    for (int i = 0; i < length; i++) {
      manual = (char)payload[i];
    }
    //    Serial.println(manual);
  }

  if (top.equals("nodemcu/device")) {
    maa = "";
    for (int i = 0; i < length; i++) {
      maa = maa + (char)payload[i];
    }
    if (maa.equals(macAdd)) {
```

```
    s = 1;
    konek = maa;
  } else {
    s = 0;
  }
}

if (s == 1) {

  Serial.println(konek);
  if (manual.equals("1")) {
    //      while(manual.equals("1")){
    //      digitalWrite(kipas, HIGH);
  }
  Serial.println("hidup");

} if (manual.equals("0")) {
  Serial.println("mati");
  //      digitalWrite(kipas, LOW);
}
if (s == 0) {
  digitalWrite(kipas, LOW);
}

if (top.equals("nodemcu/mac")) {
  Serial.println(top);
  message = "";
  for (int i = 0; i < length; i++) {
    message = message + (char)payload[i];
  }
  //    if(message.equals(mAc)){
  if (message.equals(macAdd)) {
    mcAddress = message;
    sta = 1;
    // Serial.println(mcAddress);
  }
  //    }
}

if (top.equals("nodemcu/disconnect")) {
  message = "";
  for (int i = 0; i < length; i++) {
    message = (char)payload[i];
  }
  Serial.println(message);
  dis = message;
}

if (top.equals("nodemcu/connect")) {
  message = "";
  for (int i = 0; i < length; i++) {
    message = (char)payload[i];
  }
  mess = message;
  // Serial.println(mess);
}

if (top.equals("nodemcu/dis")) {
  maa = "";
  for (int i = 0; i < length; i++) {
    maa = maa + (char)payload[i];
  }
  if (maa.equals(macAdd)) {
    disc = maa;
    Serial.println(disc);
    sta = 0;
  } else {
    sta = 1;
  }
}

if (sta == 1) {
  if (mcAddress.equals(macAdd)) {
    //    konek = macaddress;
    Serial.println(mcAddress);
    Serial.println(mess);
    conn = mess;
    if (mess.equals("")) {
      conn = 1;
    }
  }
}
if (sta == 0) {
  if (disc.equals(macAdd)) {
    Serial.println("dis: " + dis);
    //    if (dis.equals("0")) {
    Serial.println(disc);
    //      Serial.println(dis);
    conn = dis;
    //    }
  }
}

if (top.equals("nodemcu/kipas")) {
  message = "";
  Serial.println(top);
  for (int i = 0; i < length; i++) {
    message = (char)payload[i];
  }
  hasil = message.toInt();
```

```
    if (hasil == 1) {
      Serial.println("kipas hidup");
    } if (hasil  == 0) {
      Serial.println("kipas mati");
      Serial.println(hasil);
    }
  }


  if (top.equals("nodemcu/buzzer")) {
    message = "";
    Serial.println(top);
    for (int i = 0; i < length; i++) {
      message = (char)payload[i];
    }
    hasil2 = message.toInt();
    if (hasil2 == 1) {
      Serial.println("buzzer hidup");
    } else {
      Serial.println("buzzer mati");
      Serial.println(hasil2);
    }
  }
}


void setup() {
  pinMode(Sensor_Gas, INPUT);
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(buzzer, OUTPUT);
  pinMode(pinDHT, INPUT);
  pinMode(kipas, OUTPUT);
  digitalWrite(kipas, LOW);
  digitalWrite(buzzer, LOW);

  dht.begin();

  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  Serial.print("ESP32 Board MAC Address:  ");
  Serial.println(WiFi.macAddress());
  //  digitalWrite(relay, LOW);
  delay(2000);

}

void loop() {
  // put your main code here, to run repeatedly:
```

```
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  if (hasil == 1) {
    digitalWrite(kipas, HIGH);
  } if (hasil  == 0) {
    digitalWrite(kipas, LOW);
  }

  if (hasil2 == 1) {
    digitalWrite(buzzer, HIGH);
  } if (hasil2  == 0) {
    digitalWrite(buzzer, LOW);
  }

  if (konek.equals(macAdd)) {
    if (manual.equals("1")) {
      digitalWrite(kipas, HIGH);
    } else {
      digitalWrite(kipas, LOW);
    }
  }

  float gas = (((analogRead(Sensor_Gas) / 1023.00) * 100)- 80);
  int h = dht.readHumidity();
  int t = dht.readTemperature();


  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" C");
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.println(" %");
  Serial.print(gas);
  Serial.println(" PPM");


  if (conn.equals("1")) {

    if (client.publish(macTpc, String(macAdd).c_str())) {
      Serial.println("Mac Address sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(macTpc, String(macAdd).c_str());
    }

    if (client.publish(tTopic, String(t).c_str())) {
      Serial.println("Temperature sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("Temperature failed to send. Reconnecting to MQTT Broker and trying
again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(tTopic, String(t).c_str());
    }
    // PUBLISH to the MQTT Broker (topic = Humidity, defined at the beginning)
    if (client.publish(hTopic, String(h).c_str())) {
      Serial.println("Humidity sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("Humidity failed to send. Reconnecting to MQTT Broker and trying
again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(hTopic, String(h).c_str());
    }
    // PUBLISH to the MQTT Broker (topic = mq2, defined at the beginning)
    if (client.publish(gTopic, String(gas).c_str())) {
      Serial.println("mq2 sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("mq2 failed to send. Reconnecting to MQTT Broker and trying again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(gTopic, String(gas).c_str());
    }
    delay(2000);
  }
  if (conn.equals("0")) {
    Serial.println("Stop to Publish Data...");
  }

  long now = millis();
  if (now - lastMsg > 4000) {
    lastMsg = now;
    ++value;

    if (client.publish(macTopic, String(macAdd).c_str())) {
      Serial.println("Mac Address sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(macTopic, String(macAdd).c_str());
    }

    if (client.publish(temTopic, String(t).c_str())) {
      Serial.println("Temperature sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("Temperature failed to send. Reconnecting to MQTT Broker and trying
again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(temTopic, String(t).c_str());
    }
    // PUBLISH to the MQTT Broker (topic = Humidity, defined at the beginning)
    if (client.publish(humTopic, String(h).c_str())) {
      Serial.println("Humidity sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("Humidity failed to send. Reconnecting to MQTT Broker and trying
again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
```

```
      client.publish(humTopic, String(h).c_str());
    }
    // PUBLISH to the MQTT Broker (topic = mq2, defined at the beginning)
    if (client.publish(gasTopic, String(gas).c_str())) {
      Serial.println("gas sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or
not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("mq2 failed to send. Reconnecting to MQTT Broker and trying again");
      client.connect(clientID);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect
call
      client.publish(gasTopic, String(gas).c_str());
    }
  }
}
```