# LAMPIRAN

Listing Program

#include <Arduino.h>

//FirebaseESP8266.h must be included before ESP8266WiFi.h

#include "FirebaseESP8266.h"

#include <ESP8266WiFi.h>

#include <OneWire.h>

#include <DallasTemperature.h>

#include "Motor.h"

#include "Event.h"

#include "DataBase.h"

#include <ESP8266TimerInterrupt.h>

#include "Ping.h"

#define FIREBASE\_HOST "skripsi-9726b.firebaseio.com" //Without http:// or https:// schemes

#define FIREBASE\_AUTH "b5LLW3w6JPyKzecPzKwUqz3oX5sTdW2wGUOpTM4A"

#define WIFI\_SSID "ardinista"

#define WIFI\_PASSWORD "ardiasta"

#define LED  D0

#define FLOW\_SENSOR  D1

#define TRIGGER\_PIN  D3  // Arduino pin tied to trigger pin on the ultrasonic sensor.

#define ECHO\_PIN     D7  // Arduino pin tied to echo pin on the ultrasonic sensor.

#define MAX\_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

#define OVERLOAD\_PIN D6 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

#define oneWireBus   D5     // input untuk sensor DS18B20

#define MOTOR\_PIN   D2     // motor listrik

//Define data object

ESP8266Timer ITimer;

FirebaseData firebaseData1;

FirebaseData firebaseData2;

OneWire oneWire(oneWireBus);

DallasTemperature sensors (&oneWire);

Ping sonar = Ping (TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE);

Motor motor = Motor( MOTOR\_PIN, 100);

Event event = Event(1000);

Event kedip = Event(500);

DataBase flowData = DataBase();

DataBase flowAlarm = DataBase();

DataBase levelData = DataBase();

//declarasi variabel

void streamCallback(StreamData data){

  Serial.println("Stream callback ...");

  // callback motor

  Serial.print("data type = ");

  Serial.println(data.dataType());

  Serial.print("data path = ");

  Serial.println(data.dataPath());

  if (data.dataPath()=="/motor") {

    if (data.dataType() == "boolean")

    data.boolData() == 1 ? motor.motorOn(): motor.motorOff();

  }

  if (data.dataPath()=="/flow\_reset"){

    if(data.boolData() == false ){

      motor.resetFlow();

    }

  }

  if (data.dataPath()=="/sumur\_on\_level"){

    motor.setMinimumOnLevel(data.floatData());

    Serial.println(data.floatData());

  }

  if (data.dataPath()=="/sumur\_off\_level"){

    motor.setMinimumLevel(data.floatData());

    Serial.println(data.floatData());

  }

  if (data.dataType() == "json"){

    FirebaseJson &json = data.jsonObject();

    String jsonStr;

    json.toString(jsonStr, true);

    size\_t len = json.iteratorBegin();

    String key, value = "";

    int type = 0;

    for (size\_t i = 0; i < len; i++){

      json.iteratorGet(i, type, key, value);

      if (key=="motor"){

        value=="true"? motor.motorOn() : motor.motorOff();

      }

      if (key=="sumur\_on\_level"){

        motor.setMinimumOnLevel(value.toInt());

        // Serial.println(value.toInt());

      }

      if (key=="sumur\_off\_level"){

        motor.setMinimumLevel(value.toInt());

        // Serial.println(value.toInt());

      }

      if (key=="flow\_reset"){

        if (value=="false")

        {

          motor.resetFlow();

        }

      }

    }

    json.iteratorEnd();

  }

}

void streamTimeoutCallback(bool timeout)

{

  if (timeout)

  {

    Serial.println();

    Serial.println("Stream timeout, resume streaming...");

    Serial.println();

  }

}

float flowValue;

unsigned long counter;

void ICACHE\_RAM\_ATTR  flowCallback(){

  static int tambah;

  if (digitalRead(FLOW\_SENSOR)==HIGH){

    if (tambah == 1){

      counter++;

      tambah = 0;

    }

  }

  else{

    tambah=1;

  }

}

void ICACHE\_RAM\_ATTR TimerHandler(){

  noInterrupts(); // again

  flowValue=counter/3.9;

  counter=0;

  interrupts();

}

void setup(){

    // Interval in microsecs

  ITimer.attachInterruptInterval(1000000, TimerHandler);

  Serial.begin(9600);

  WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD);

  Serial.print("Connecting to Wi-Fi");

  while (WiFi.status() != WL\_CONNECTED)

  {

    Serial.print(".");

    delay(300);

  }

  Serial.println();

  Serial.print("Connected with IP: ");

  Serial.println(WiFi.localIP());

  Serial.println();

  Firebase.begin(FIREBASE\_HOST, FIREBASE\_AUTH);

  Firebase.reconnectWiFi(true);

  //Set the size of WiFi rx/tx buffers in the case where we want to work with large data.

  firebaseData1.setBSSLBufferSize(1024, 1024);

  //Set the size of HTTP response buffers in the case where we want to work with large data.

  firebaseData1.setResponseSize(1024);

  // Set the size of WiFi rx/tx buffers in the case where we want to work with large data.

  firebaseData2.setBSSLBufferSize(1024, 1024);

  // Set the size of HTTP response buffers in the case where we want to work with large data.

  firebaseData2.setResponseSize(1024);

  if (!Firebase.beginStream(firebaseData1, "/nodeGet"))

  {

    Serial.println("------------------------------------");

    Serial.println("Can't begin stream connection...");

    Serial.println("REASON: " + firebaseData1.errorReason());

    Serial.println("------------------------------------");

    Serial.println();

  }

  Firebase.setStreamCallback(firebaseData1, streamCallback, streamTimeoutCallback);

  Serial.println("start.....");

  attachInterrupt(FLOW\_SENSOR, flowCallback, CHANGE);

  pinMode(FLOW\_SENSOR,INPUT\_PULLUP);

  pinMode(OVERLOAD\_PIN,INPUT\_PULLUP);

  pinMode(LED,OUTPUT);

  sonar.setMirorLenght(21.8);

}

void loop(){

  float level =  sonar.getMirorDistance();

  if (kedip.getEvent())

  {

    digitalRead(LED)==LOW?

    digitalWrite(D0,HIGH):

    digitalWrite(D0,LOW);

  }

  if (event.getEvent()) {

    //set flow ke firebasw

    if (flowData.setData(flowValue)){

      Firebase.setDouble(firebaseData2, "/nodeSet/flow",(flowValue));

    }

    //cek alarm flow sensor

    if (!motor.setFlow(flowValue)){

      Firebase.setBool(firebaseData2, "/alarm/flow",true);

      if (flowAlarm.setData(HIGH)){

        Firebase.setBool(firebaseData2, "/nodeGet/flow\_reset",true);

      }

    }

    else{

      flowAlarm.setData(LOW);

      Firebase.setBool(firebaseData2, "/alarm/flow",false);

    }

    // Sensor suhu pada motor listrik

    static float motorTemp;

    sensors.requestTemperatures();

    if (motorTemp != sensors.getTempCByIndex(0)){

      motor.setTemperature(sensors.getTempCByIndex(0));

      if (Firebase.setDouble(firebaseData2, "/nodeSet/suhu",sensors.getTempCByIndex(0))){

        if (sensors.getTempCByIndex(0)>=100){

          Firebase.setBool(firebaseData2, "/alarm/overheat",true);

        }

        else{

          Firebase.setBool(firebaseData2, "/alarm/overheat",false);

        }

      }

    }

    //sensor overload pada motor

    if (digitalRead(OVERLOAD\_PIN)==LOW ) {

      motor.setOverload(true);

      Firebase.setBool(firebaseData2, "/alarm/overload",true);

    }

    else {

      motor.setOverload(false);

      Firebase.setBool(firebaseData2, "/alarm/overload",false);

    }

    if (motor.setLevel(level)) {

      Firebase.setBool(firebaseData2, "/alarm/level",false);

    }

    else {

      Firebase.setBool(firebaseData2, "/alarm/level",true);

    }

    if (levelData.setData(level))

    {

      Firebase.setDouble(firebaseData2, "/nodeSet/sumur",(level));

      Firebase.setDouble(firebaseData2, "/tankGet/sumur",(level));

    }

  }

}

Database cpp

#include "DataBase.h"

bool DataBase::setData(String data){

    this->dataString =data;

    if (this->dataString != this->tempDataString ){

        this->dataUpdate = true;

        this->tempDataString = data;

    }

    else{

        this->dataUpdate = false;

    }

    return this->dataUpdate;

}

bool DataBase::setData(bool data){

    this->dataBool =data;

    if (this->dataBool != this->tempDataBool ){

        this->dataUpdate = true;

        this->tempDataBool = data;

    }

    else{

        this->dataUpdate = false;

    }

    return this->dataUpdate;

}

bool DataBase::setData(int data){

    this->dataInt=data;

    if (this->dataInt != this->tempDataInt ){

        this->dataUpdate = true;

        this->tempDataInt = data;

    }

    else{

        this->dataUpdate = false;

    }

    return this->dataUpdate;

}

bool DataBase::setData(float data){

    this->dataFloat = data;

    if (this->dataFloat != this->tempDataFloat ){

        this->dataUpdate = true;

        this->tempDataFloat = data;

    }

    else{

        this->dataUpdate = false;

    }

    return this->dataUpdate;

}

Event cpp

#include "Event.h"

bool Event::getEvent(){

    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval) {

        previousMillis = currentMillis;

        return true;

    }

    else return false;

}

void Event::setEvent(int milli\_second){

    this->interval = milli\_second;

}

Event::Event(int milli\_second)

{

    this->interval = milli\_second;

    this->previousMillis=millis();

}

Motor cpp

#include "Motor.h"

void Motor::setOverload( bool ol)

{

    this->overload = ol;

    if (this->overload == false )

    {

        this->turnOn();

    }

    if (this->motorState == HIGH &&  this->overload == true)

    {

        this->turnOff();

    }

}

void Motor::setMinimumOnLevel(float temp){

    this->minimumOnLevel=temp;

}

void Motor::setMinimumLevel( float cm)

{

    this->minimumLevel = cm;

    if (this->level > this->minimumLevel)

    {

        this->turnOn();

    }

    if (this->motorState == HIGH &&  this->level < this->minimumLevel)

    {

        this->turnOff();

    }

}

int Motor::setLevel( float cm)

{

    this->level = cm;

    if (this->level >= this->minimumOnLevel)

    {

        this->levelState=HIGH;

        this->turnOn();

    }

    if (this->motorState == HIGH && this->level  <= this->minimumLevel)

    {

        this->levelState=LOW;

        this->turnOff();

    }

    return this->levelState;

}

int Motor::setFlow( float flow){

    if (digitalRead(this->motorPin) == HIGH &&  flow == 0)

    {

        if ((millis()-this->lastOn)>2000L)

        {

            this->flowReset=HIGH;

            this->turnOn();

            // return 0;

        }

    }

    return !this->flowReset;

}

void Motor::resetFlow(){

    this->flowReset=LOW;

    this->turnOn();

}

void Motor::motorOn(){

    this->motorState=HIGH;

    this->turnOn();

}

void Motor::turnOn(){

    if (this->motorTempeerature >= this->maxTemperature) {

        this->turnOff();

        return;

    }

    if (this->level <= this->minimumLevel) {

        this->turnOff();

        return;

    }

    if (this->flowReset == HIGH) {

        this->turnOff();

        return;

    }

    if (this->overload == true) {

        this->turnOff();

        return;

    }

    if (this->levelState == LOW) {

        this->turnOff();

        return;

    }

    if (digitalRead(this->motorPin)!=this->motorState )

    {

        this->lastOn=millis();

    }

    digitalWrite(this->motorPin,this->motorState);  // motor listrik

}

void Motor::turnOff()

{

    digitalWrite(this->motorPin,LOW);    // motor listrik

}

void Motor::motorOff()

{

    this->motorState=LOW;

    this->turnOff();

}

void Motor::setTemperature( float temp)

{

    this->motorTempeerature = temp;

    temp >= this->maxTemperature?

        this->turnOff():

        this->turnOn();

}

Motor::Motor(int motor\_pin,  float max\_temperature)

{

    this->motorPin=motor\_pin;

    this->maxTemperature=max\_temperature;

    this->minimumLevel=2;

    this->minimumOnLevel=4;

    this->level=10;

    this->motorTempeerature=30;

    this->overload = false;

    this->motorState = LOW;

    this->levelState = HIGH;

    pinMode(this->motorPin,OUTPUT);

}

Motor::~Motor()

{

}

Ping cpp

#include "Ping.h"

void Ping::setMirorLenght(float lenght){

    this->mirror=lenght;

}

float Ping::getMirorDistance(){

    SUM = SUM - READINGS[INDEX];                        // Remove the oldest entry from the sum

    VALUE = (this->mirror -(this->sonar->ping()/57.0)); // Read the next sensor value

    READINGS[INDEX] = VALUE;                            // Add the newest reading to the window

    SUM = SUM + VALUE;                                  // Add the newest reading to the sum

    INDEX = (INDEX+1) % WINDOW\_SIZE;                    // Increment the index, and wrap to 0 if it exceeds the window size

    AVERAGED = SUM / WINDOW\_SIZE;                       // Divide the sum of the window by the window size for the result

    return AVERAGED;

}

Ping::Ping(int triger, int echo, int max\_distance)

{

    this->sonar = new NewPing (triger, echo, max\_distance);

}