**DOKUMEN DESAIN PERANGKAT LUNAK**

**SISTEM MONITORING DAYA LISTRIK**

SOURCE CODE

*LSKK-SW-SC-01-SMDL*



**PT. LANGGENG SEJAHTERA KREASI KOMPUTASI**

**©2019**

Dokumen Desain Perangkat Lunak, MONITORING DAYA LISTRIK

Source Code Sistem Monitoring Daya Listrik

PT. Langgeng Sejahtera Kreasi Komputasi ©2019

# Deskripsi Aplikasi

# Sistem monitoring daya listrik merupakan sistem *Remote Metering yang* dibangun menggunakan konsep IOT sehingga dapat di implementasi dengan jumlah yang banyak dengan fitur yang semakin kompleks dengan cepat.

# Penggunaan saat intensitas yang tinggi pada skala yang banyak memungkinkan berbagai hal yang tidak dapat dihitung sebelumnya:

# Estimasi status perangkat terpasang dengan granularisasi yang tinggi hingga bahkan dapat memperkirakan terjadinya abnormalitas;

1. Prediksi *demand* yang semakin akurat sehingga *supply* dapat dibuat sangat dekat dengan demand dll. Semua tentunya memanfaatkan teknologi *big-data* dan AI.

# *Source Code*

Berikut *source code* untuk *microcontroller* .

|  |
| --- |
| /\*  Develop by : Nurman Hariyanto  Email : nurman.hariyanto13@gmail.com  Project : Homeautomation  Version : 3.0  \*/  //library  #include <FS.h> //this needs to be first, or it all crashes and burns...  #include <Ticker.h>  #include <ESP8266WiFi.h>  #include <DNSServer.h>  #include <ESP8266WebServer.h>  #include <WiFiManager.h>  #include <ArduinoJson.h>  //#include <SPI.h>  //#include "MFRC522.h"  #include <PubSubClient.h>  #include "config.h"  #include <SoftwareSerial.h> //import SoftwareSerial library  #include <SDM.h> //import SDM library  SoftwareSerial swSerSDM(D2, D3); //config SoftwareSerial (rx->D7 / tx->D8)  SDM sdm(swSerSDM, 9600, D5); //config SDM  String statusKWH = "A0";  int ssrPIN = D6;  /\*  Setup pin and load config file  \*/  void setup() {  pinMode(D6, OUTPUT);  digitalWrite(D6, LOW);  Serial.begin(9600);  Serial.println(F("Booting...."));  //read config wifi,mqtt dan yang lain  ReadConfigFile();  setup\_wifi();  SaveConfigFile();  client.setServer(mqtt\_server, mqtt\_port); //set mqtt server  client.setCallback(callback);  sdm.begin(); //initialize sdm communication  Serial.println("reseto");  }  //setup wifi  void setup\_wifi() {  delay(10);  WiFi.macAddress(MAC\_array);  for (int i = 0; i < sizeof(MAC\_array) - 1; ++i) {  sprintf(MAC\_char, "%s%02x:", MAC\_char, MAC\_array[i]);  }  sprintf(MAC\_char, "%s%02x", MAC\_char, MAC\_array[sizeof(MAC\_array) - 1]);  WiFiManagerParameter custom\_mqtt\_server("server", "mqtt server", mqtt\_server, 40);  String(mqtt\_port).toCharArray(smqtt\_port, 5);  WiFiManagerParameter custom\_mqtt\_port("port", "mqtt port", smqtt\_port, 5);  WiFiManagerParameter custom\_mqtt\_user("user", "mqtt user", mqtt\_user, 40);  WiFiManagerParameter custom\_mqtt\_password("password", "mqtt password", mqtt\_password, 40);  WiFiManagerParameter custom\_mqtt\_keywords1("keyword1", "mqtt keyword1", mqtt\_keywords1, 40);  WiFiManagerParameter custom\_mqtt\_keywords2("keyword2", "mqtt keyword2", mqtt\_keywords2, 40);  WiFiManagerParameter custom\_mqtt\_keywords3("keyword3", "mqtt keyword3", mqtt\_keywords3, 40);  WiFiManagerParameter custom\_mqtt\_keywords4("keyword4", "mqtt keyword4", mqtt\_keywords4, 40);  WiFiManager wifiManager;  wifiManager.setSaveConfigCallback(saveConfigCallback);  wifiManager.addParameter( & custom\_mqtt\_server);  wifiManager.addParameter( & custom\_mqtt\_port);  wifiManager.addParameter( & custom\_mqtt\_user);  wifiManager.addParameter( & custom\_mqtt\_password);  wifiManager.addParameter(&custom\_mqtt\_keywords1);  wifiManager.addParameter(&custom\_mqtt\_keywords2);  wifiManager.addParameter(&custom\_mqtt\_keywords3);  wifiManager.addParameter(&custom\_mqtt\_keywords4);  //fetches ssid and pass and tries to connect  //if it does not connect it starts an access point with the specified name  //here "AutoConnectAP"  //and goes into a blocking loop awaiting configuration  if (!wifiManager.autoConnect(MAC\_char, "password")) {  Serial.println("failed to connect and hit timeout");  delay(2000);  //reset and try again, or maybe put it to deep sleep  ESP.reset();  delay(2000);  }  //print local ip if connected  Serial.println("");  Serial.println("WiFi connected");  Serial.println("IP address: ");  Serial.println(WiFi.localIP());  }  String mac2String(byte ar[]) {  String s;  for (byte i = 0; i < 6; ++i)  {  char buf[3];  sprintf(buf, "%2X", ar[i]);  s += buf;  if (i < 5) s += ':';  }  return s;  }  void printMACAddress() {  WiFi.macAddress(mac);  MACAddress = mac2String(mac);  Serial.println(MACAddress);  }  //calback  void callback(char \* topic, byte \* payload, unsigned int length) {  Serial.print("Message arrived [");  Serial.print(topic);  Serial.print("] ");  Serial.println();  if (!strncmp((char \*)payload, "A1", length)) {  digitalWrite(ssrPIN, HIGH);  statusKWH = "A1";  }  if (!strncmp((char \*)payload, "A0", length)) {  digitalWrite(ssrPIN, LOW);  statusKWH = "A0";  }  String dataControl = String(statusKWH);  char datastatSSR[20];  dataControl.toCharArray(datastatSSR, sizeof(datastatSSR));  Serial.println(datastatSSR);  if (client.publish(mqtt\_keywords3, datastatSSR) == true) {  Serial.println("Success sending message status relay");  Serial.println(datastatSSR);  } else {  Serial.println("Error sending message");  }  //inset here if want to control microcontroller  }  void reconnect() {  // Loop until we're reconnected  printMACAddress();  const char\* CL;  CL = MACAddress.c\_str();  Serial.println(CL);  while (!client.connected()) {  Serial.print("Attempting MQTT connection...%s");  Serial.println(mqtt\_server);  // Attempt to connect  if (client.connect(MAC\_char, mqtt\_user, mqtt\_password)) {  Serial.println("connected");  // Once connected, publish an announcement...  //client.publish(mqtt\_keywords1, "halo smarthome");  client.subscribe(mqtt\_keywords2);  } else {  Serial.print("failed, rc=");  Serial.print(client.state());  if (client.state() == 4) ESP.restart();  else {  Serial.println(" try again in 5 seconds");  // Wait 5 seconds before retrying  //delay(10000);  }  }  }  }  //loop publish  void loop() {  if (!client.connected()) {  reconnect();  }  client.loop();  String pubmsg = "";    String jenis\_iot = "Input-Sensor-KWHmeter";  String mac = MACAddress;  float data\_1 = (sdm.readVal(SDM220T\_VOLTAGE));  float data\_2 = (sdm.readVal(SDM220T\_CURRENT));  float data\_3 = (sdm.readVal(SDM220T\_POWER));  String kode\_device = "EDA997102A";  Serial.print("Publish message: ");  pubmsg = "#" + jenis\_iot + "#" + mac + "#" + data\_1 + "#" + data\_2 + "#" + data\_3 + "#" + kode\_device;  Serial.println(pubmsg);  client.publish(mqtt\_keywords4, pubmsg.c\_str());  //data sdm 220  String dataEastron = String(sdm.readVal(SDM220T\_VOLTAGE)) + "#" + (sdm.readVal(SDM220T\_CURRENT)) + "#" + (sdm.readVal(SDM220T\_POWER)) + "#" + (sdm.readVal(SDM220T\_ACTIVE\_APPARENT\_POWER)) + "#" +  (sdm.readVal(SDM220T\_REACTIVE\_APPARENT\_POWER)) + "#" + (sdm.readVal(SDM220T\_POWER\_FACTOR)) + "#" + (sdm.readVal(SDM220T\_PHASE\_ANGLE)) + "#" + (sdm.readVal(SDM220T\_FREQUENCY)) + "#" +  (sdm.readVal(SDM220T\_IMPORT\_ACTIVE\_ENERGY)) + "#" + (sdm.readVal(SDM220T\_EXPORT\_ACTIVE\_ENERGY)) + "#" + (sdm.readVal(SDM220T\_IMPORT\_REACTIVE\_ENERGY)) + "#" + (sdm.readVal(SDM220T\_EXPORT\_REACTIVE\_ENERGY)) + "#" +  (sdm.readVal(SDM220T\_TOTAL\_ACTIVE\_ENERGY)) + "#" + (sdm.readVal(SDM220T\_TOTAL\_REACTIVE\_ENERGY)) + "#";  char dataSendEastron[200];  dataEastron.toCharArray(dataSendEastron, sizeof(dataSendEastron));  Serial.println(dataSendEastron);  client.publish(mqtt\_keywords1, dataSendEastron);  Serial.println("Success sending message to: ");  Serial.print(mqtt\_keywords1);  delay(5000);  } |

Source 1. Main *source code microcontroller*

|  |
| --- |
| /\* Library for reading SDM 120/220/230/630 Modbus Energy meters.  \* Reading via Hardware or Software Serial library & rs232<->rs485 converter  \* 2016-2018 Reaper7 (tested on wemos d1 mini->ESP8266 with Arduino 1.9.0-beta & 2.4.1 esp8266 core)  \* crc calculation by Jaime García (https://github.com/peninquen/Modbus-Energy-Monitor-Arduino/)  \*/  //------------------------------------------------------------------------------  #ifndef SDM\_h  #define SDM\_h  //------------------------------------------------------------------------------  #include <Arduino.h>  #include <SDM\_Config\_User.h>  #ifdef USE\_HARDWARESERIAL  #include <HardwareSerial.h>  #else  #include <SoftwareSerial.h>  #endif  //------------------------------------------------------------------------------  //DEFAULT CONFIG (DO NOT CHANGE ANYTHING!!! for changes use SDM\_Config\_User.h):  //------------------------------------------------------------------------------  #ifndef SDM\_UART\_BAUD  #define SDM\_UART\_BAUD 9600 //default baudrate  #endif  #ifndef DERE\_PIN  #define DERE\_PIN NOT\_A\_PIN //default digital pin for control MAX485 DE/RE lines (connect DE & /RE together to this pin)  #endif  #ifdef USE\_HARDWARESERIAL  #ifndef SDM\_UART\_CONFIG  #define SDM\_UART\_CONFIG SERIAL\_8N1 //default hardware uart config  #endif  #ifndef SWAPHWSERIAL  #define SWAPHWSERIAL 0 //(only esp8266) when hwserial used, then swap uart pins from 3/1 to 13/15 (default not swap)  #endif  #endif  #ifndef MAX\_MILLIS\_TO\_WAIT  #define MAX\_MILLIS\_TO\_WAIT 500 //default max time to wait for response from SDM  #endif  //------------------------------------------------------------------------------  #define FRAMESIZE 9 //size of out/in array  #define SDM\_REPLY\_BYTE\_COUNT 0x04 //number of bytes with data  #define SDM\_B\_01 0x01 //BYTE 1 -> slave address (default value 1 read from node 1)  #define SDM\_B\_02 0x04 //BYTE 2 -> function code (default value 0x04 read from 3X input registers)  //BYTES 3 & 4 (BELOW)  //SDM 120 registers  #define SDM120C\_VOLTAGE 0x0000 //V  #define SDM120C\_CURRENT 0x0006 //A  #define SDM120C\_POWER 0x000C //W  #define SDM120C\_ACTIVE\_APPARENT\_POWER 0x0012 //VA  #define SDM120C\_REACTIVE\_APPARENT\_POWER 0x0018 //VAR  #define SDM120C\_POWER\_FACTOR 0x001E //  #define SDM120C\_FREQUENCY 0x0046 //Hz  #define SDM120C\_IMPORT\_ACTIVE\_ENERGY 0x0048 //Wh  #define SDM120C\_EXPORT\_ACTIVE\_ENERGY 0x004A //Wh  #define SDM120C\_TOTAL\_ACTIVE\_ENERGY 0x0156 //Wh  //SDM 220 registers  #define SDM220T\_VOLTAGE 0x0000 //V  #define SDM220T\_CURRENT 0x0006 //A  #define SDM220T\_POWER 0x000C //W  #define SDM220T\_ACTIVE\_APPARENT\_POWER 0x0012 //VA  #define SDM220T\_REACTIVE\_APPARENT\_POWER 0x0018 //VAR  #define SDM220T\_POWER\_FACTOR 0x001E //  #define SDM220T\_PHASE\_ANGLE 0x0024 //DEGREE  #define SDM220T\_FREQUENCY 0x0046 //Hz  #define SDM220T\_IMPORT\_ACTIVE\_ENERGY 0x0048 //Wh  #define SDM220T\_EXPORT\_ACTIVE\_ENERGY 0x004A //Wh  #define SDM220T\_IMPORT\_REACTIVE\_ENERGY 0x004C //VARh  #define SDM220T\_EXPORT\_REACTIVE\_ENERGY 0x004E //VARh  #define SDM220T\_TOTAL\_ACTIVE\_ENERGY 0x0156 //Wh  #define SDM220T\_TOTAL\_REACTIVE\_ENERGY 0x0158 //VARh  //SDM 230 registers  #define SDM230\_VOLTAGE 0x0000 //V  #define SDM230\_CURRENT 0x0006 //A  #define SDM230\_POWER 0x000C //W  #define SDM230\_ACTIVE\_APPARENT\_POWER 0x0012 //VA  #define SDM230\_REACTIVE\_APPARENT\_POWER 0x0018 //VAR  #define SDM230\_POWER\_FACTOR 0x001E //  #define SDM230\_PHASE\_ANGLE 0x0024 //DEGREE  #define SDM230\_FREQUENCY 0x0046 //Hz  #define SDM230\_IMPORT\_ACTIVE\_ENERGY 0x0048 //Wh  #define SDM230\_EXPORT\_ACTIVE\_ENERGY 0x004A //Wh  #define SDM230\_IMPORT\_REACTIVE\_ENERGY 0x004C //VARh  #define SDM230\_EXPORT\_REACTIVE\_ENERGY 0x004E //VARh  #define SDM230\_TOTAL\_SYSTEM\_POWER\_DEMAND 0x0054 //W  #define SDM230\_MAXIMUM\_SYSTEM\_POWER\_DEMAND 0x0056 //W  #define SDM230\_CURRENT\_POSITIVE\_POWER\_DEMAND 0x0058 //W  #define SDM230\_MAXIMUM\_POSITIVE\_POWER\_DEMAND 0x005A //W  #define SDM230\_CURRENT\_REVERSE\_POWER\_DEMAND 0x005C //W  #define SDM230\_MAXIMUM\_REVERSE\_POWER\_DEMAND 0x005E //W  #define SDM230\_CURRENT\_DEMAND 0x0102 //Amps  #define SDM230\_MAXIMUM\_CURRENT\_DEMAND 0x0108 //Amps  #define SDM230\_TOTAL\_ACTIVE\_ENERGY 0x0156 //kwh  #define SDM230\_TOTAL\_REACTIVE\_ENERGY 0x0158 //kvarh  #define SDM230\_CURRENT\_RESETTABLE\_TOTAL\_ACTIVE\_ENERGY 0x0180 //Wh  #define SDM230\_CURRENT\_RESETTABLE\_TOTAL\_REACTIVE\_ENERGY 0x0182 //VARh  //SDM 630 registers  #define SDM630\_VOLTAGE1 0x0000 //V  #define SDM630\_VOLTAGE2 0x0002 //V  #define SDM630\_VOLTAGE3 0x0004 //V  #define SDM630\_CURRENT1 0x0006 //A  #define SDM630\_CURRENT2 0x0008 //A  #define SDM630\_CURRENT3 0x000A //A  #define SDM630\_CURRENTSUM 0x0030 //A  #define SDM630\_POWER1 0x000C //W  #define SDM630\_POWER2 0x000E //W  #define SDM630\_POWER3 0x0010 //W  #define SDM630\_POWERTOTAL 0x0034 //W  #define SDM630\_VOLT\_AMPS1 0x0012 //VA  #define SDM630\_VOLT\_AMPS2 0x0014 //VA  #define SDM630\_VOLT\_AMPS3 0x0016 //VA  #define SDM630\_VOLT\_AMPS\_TOTAL 0x0038 //VA  #define SDM630\_VOLT\_AMPS\_REACTIVE1 0x0018 //VAr  #define SDM630\_VOLT\_AMPS\_REACTIVE2 0x001A //VAr  #define SDM630\_VOLT\_AMPS\_REACTIVE3 0x001C //VAr  #define SDM630\_VOLT\_AMPS\_REACTIVE\_TOTAL 0x003C //VAr  #define SDM630\_POWER\_FACTOR1 0x001E  #define SDM630\_POWER\_FACTOR2 0x0020  #define SDM630\_POWER\_FACTOR3 0x0022  #define SDM630\_POWER\_FACTOR\_TOTAL 0x003E  #define SDM630\_PHASE\_ANGLE1 0x0024 //Degrees  #define SDM630\_PHASE\_ANGLE2 0x0026 //Degrees  #define SDM630\_PHASE\_ANGLE3 0x0028 //Degrees  #define SDM630\_PHASE\_ANGLE\_TOTAL 0x0042 //Degrees  #define SDM630\_VOLTAGE\_AVERAGE 0x002A //V  #define SDM630\_CURRENT\_AVERAGE 0x002E //A  #define SDM630\_FREQUENCY 0x0046 //HZ  #define SDM630\_IMPORT\_ACTIVE\_ENERGY 0x0048 //Wh  #define SDM630\_EXPORT\_ACTIVE\_ENERGY 0x004A //Wh  #define SDM630\_IMPORT\_REACTIVE\_ENERGY 0x004C //VARh  #define SDM630\_EXPORT\_REACTIVE\_ENERGY 0x004E //VARh  #define SDM630\_TOTAL\_SYSTEM\_POWER\_DEMAND 0x0054 //W  #define SDM630\_MAXIMUM\_TOTAL\_SYSTEM\_POWER 0x0056 //W  #define SDM630\_PHASE\_1\_LN\_VOLTS\_THD 0x00EA //%  #define SDM630\_PHASE\_2\_LN\_VOLTS\_THD 0x00EC //%  #define SDM630\_PHASE\_3\_LN\_VOLTS\_THD 0x00EE //%  #define SDM630\_AVERAGE\_VOLTS\_THD 0x00F8 //%  #define SDM630\_PHASE\_1\_CURRENT\_THD 0x00F0 //%  #define SDM630\_PHASE\_2\_CURRENT\_THD 0x00F2 //%  #define SDM630\_PHASE\_3\_CURRENT\_THD 0x00F4 //%  #define SDM630\_AVERAGE\_CURRENT\_THD 0x00FA //%  #define SDM630\_IMPORT1 0x015a //kWh  #define SDM630\_IMPORT2 0x015c //kWh  #define SDM630\_IMPORT3 0x015e //kWh  #define SDM630\_EXPORT1 0x0160 //kWh  #define SDM630\_EXPORT2 0x0162 //kWh  #define SDM630\_EXPORT3 0x0164 //kWh  #define SDM630\_TOTAL\_ENERGY1 0x0166 //kWh  #define SDM630\_TOTAL\_ENERGY2 0x0168 //kWh  #define SDM630\_TOTAL\_ENERGY3 0x016a //kWh  #define SDM\_B\_05 0x00 //BYTE 5  #define SDM\_B\_06 0x02 //BYTE 6  //------------------------------------------------------------------------------  #define SDM\_ERR\_NO\_ERROR 0 //no error  #define SDM\_ERR\_CRC\_ERROR 1 //crc error  #define SDM\_ERR\_WRONG\_BYTES 2 //bytes b0,b1 or b2 wrong  #define SDM\_ERR\_NOT\_ENOUGHT\_BYTES 3 //not enough bytes from sdm  #define SDM\_ERR\_TIMEOUT 4 //timeout  //------------------------------------------------------------------------------  class SDM {  public:  #ifdef USE\_HARDWARESERIAL  SDM(HardwareSerial& serial, long baud = SDM\_UART\_BAUD, int dere\_pin = DERE\_PIN, int config = SDM\_UART\_CONFIG, bool swapuart = SWAPHWSERIAL);  #else  SDM(SoftwareSerial& serial, long baud = SDM\_UART\_BAUD, int dere\_pin = DERE\_PIN);  #endif  virtual ~SDM();  void begin(void);  float readVal(uint16\_t reg, uint8\_t node = SDM\_B\_01); //read value from register = reg and from deviceId = node  uint16\_t getErrCode(bool \_clear = false); //return last errorcode (optional clear this value, default flase)  uint16\_t getErrCount(bool \_clear = false); //return total errors count (optional clear this value, default flase)  void clearErrCode(); //clear last errorcode  void clearErrCount(); //clear total errors count  private:  #ifdef USE\_HARDWARESERIAL  HardwareSerial& sdmSer;  #else  SoftwareSerial& sdmSer;  #endif  #ifdef USE\_HARDWARESERIAL  int \_config = SDM\_UART\_CONFIG;  bool \_swapuart = SWAPHWSERIAL;  #endif  long \_baud = SDM\_UART\_BAUD;  int \_dere\_pin = DERE\_PIN;  uint16\_t readingerrcode = SDM\_ERR\_NO\_ERROR; //4 = timeout; 3 = not enough bytes; 2 = number of bytes OK but bytes b0,b1 or b2 wrong, 1 = crc error  uint16\_t readingerrcount = 0; //total errors couter  uint16\_t calculateCRC(uint8\_t \*array, uint8\_t num);  };  #endif //SDM\_h |

Source 2. *Header* file SDM

|  |
| --- |
| /\* Library for reading SDM 120/220/230/630 Modbus Energy meters.  \* Reading via Hardware or Software Serial library & rs232<->rs485 converter  \* 2016-2019 Reaper7 (tested on wemos d1 mini->ESP8266 with Arduino 1.9.0-beta & 2.4.1 esp8266 core)  \* crc calculation by Jaime Garc�a (https://github.com/peninquen/Modbus-Energy-Monitor-Arduino/)  \*/  /\*  \* USER CONFIG:  \*/  //------------------------------------------------------------------------------  /\*  \* define or undefine USE\_HARDWARESERIAL (uncomment only one or none)  \*/  //#undef USE\_HARDWARESERIAL  //#define USE\_HARDWARESERIAL  //------------------------------------------------------------------------------  /\*  \* define user baudrate  \*/  //#define SDM\_UART\_BAUD 9600  //------------------------------------------------------------------------------  /\*  \* define user DERE\_PIN for control MAX485 DE/RE lines (connect DE & /RE together to this pin)  \*/  //#define DERE\_PIN NOT\_A\_PIN  //------------------------------------------------------------------------------  #ifdef USE\_HARDWARESERIAL  /\*  \* define user SDM\_UART\_CONFIG  \*/  //#define SDM\_UART\_CONFIG SERIAL\_8N1  //----------------------------------------------------------------------------  /\*  \* define user SWAPHWSERIAL, if true(1) then swap uart pins from 3/1 to 13/15 (only ESP8266)  \*/  //#define SWAPHWSERIAL 0  #endif  //------------------------------------------------------------------------------  /\*  \* define user MAX\_MILLIS\_TO\_WAIT to wait for response from SDM  \*/  //#define MAX\_MILLIS\_TO\_WAIT 500  //------------------------------------------------------------------------------ |

Source 3. *Header* file SDM *user config*

|  |
| --- |
| #ifndef \_\_CONFIGFILE\_\_  #define \_\_CONFIGFILE\_\_  //flag for saving data  bool shouldSaveConfig = false;  // Update these with values suitable for your network.  char wifi\_ssid[34] = "LSKK\_AP2.1";  char wifi\_password[34] = "lskk@ppejuang";  char mqtt\_server[40] = "rmqemeter.pptik.id";  int mqtt\_port = 1883;  char smqtt\_port[5] = "1883";  char mqtt\_user[40] = "/kwhmeter:kwhmeter";  char mqtt\_password[40] = "!!\_kwhmeter";  char mqtt\_keywords1[40] = "datakwh";  char mqtt\_keywords2[40] = "controlssr1";  char mqtt\_keywords3[40] = "statusssr1";  char mqtt\_keywords4[40] = "kwh\_inside";  const long interval = 2000;  uint8\_t MAC\_array[6];  char MAC\_char[18];  WiFiClient espClient;  PubSubClient client(espClient);  byte mac[6];  String MACAddress;  void SaveConfigCallback ();  void ReadConfigFile();  #endif |

Source 4. *Config* *microcontroller*

|  |
| --- |
| #include "config.h"  //callback notifying us of the need to save config  void saveConfigCallback () {  Serial.println("Should save config");  shouldSaveConfig = true;  }  void ReadConfigFile()  {  //read configuration from FS json  Serial.println("mounting FS...");  if (SPIFFS.begin()) {  Serial.println("mounted file system");  if (SPIFFS.exists("/config.json")) {  //SPIFFS.remove("/config.json");  //file exists, reading and loading  Serial.println("reading config file");  File configFile = SPIFFS.open("/config.json", "r");  if (configFile) {  Serial.println("opened config file");  size\_t size = configFile.size();  // Allocate a buffer to store contents of the file.  std::unique\_ptr<char[]> buf(new char[size]);  configFile.readBytes(buf.get(), size);  DynamicJsonBuffer jsonBuffer;  JsonObject& json = jsonBuffer.parseObject(buf.get());  json.printTo(Serial);  if (json.success()) {  Serial.println("\nparsed json");  strcpy(mqtt\_server, json["mqtt\_server"]);  String port = json["mqtt\_port"];  mqtt\_port = port.toInt();  strcpy(mqtt\_user, json["mqtt\_user"]);  strcpy(mqtt\_password, json["mqtt\_password"]);  strcpy(mqtt\_keywords1, json["mqtt\_keywords1"]);  strcpy(mqtt\_keywords2, json["mqtt\_keywords2"]);  strcpy(mqtt\_keywords3, json["mqtt\_keywords3"]);  strcpy(mqtt\_keywords3, json["mqtt\_keywords4"]);  } else {  Serial.println("failed to load json config");  }  }  }  } else {  Serial.println("failed to mount FS");  }  //end read  }  String IpAddress2String(const IPAddress& ipAddress)  {  return String(ipAddress[0]) + String(".") + \  String(ipAddress[1]) + String(".") + \  String(ipAddress[2]) + String(".") + \  String(ipAddress[3]) ;  }  void SaveConfigFile()  {  //save the custom parameters to FS  if (shouldSaveConfig) {  Serial.println("saving config");  DynamicJsonBuffer jsonBuffer;  JsonObject& json = jsonBuffer.createObject();  json["mqtt\_server"] = mqtt\_server;  json["mqtt\_port"] = String(mqtt\_port);  json["mqtt\_user"] = mqtt\_user;  json["mqtt\_password"] = mqtt\_password;  json["mqtt\_keywords1"] = mqtt\_keywords1;  File configFile = SPIFFS.open("/config.json", "w");  if (!configFile) {  Serial.println("failed to open config file for writing");  }  json.printTo(Serial);  json.printTo(configFile);  configFile.close();  //end save  }  }  void ResetSettings()  {  WiFiManager wifiManager;  wifiManager.resetSettings();  delay(2000);  ESP.restart();  } |

Source 4. *Config* file *microcontroller*

# *Release Notes*

Berikut *release notes* aplikasi untuk sistem monitoring daya listrik.

### *SMDL-SC, b1903, v1.0*

* Versi rilis pertama

# *Approval*

Dibuat oleh Tim *S/W* *Engineer* pada 13 Desember 2019.

Dicek dan disetujui pada 13 Desember 2019, oleh:

Mochamad Vicky Ghani Aziz