9906

```
#include <stdio.h>
#include <WiFi.h>
#include <MQTT.h>
#include <SPI.h>
#include "DallasTemperature.h"
#include "EmonLib.h"
#include <PubSubClient.h>
#include <time.h>
#include <ArduinoJson.h>
#include "ESPDateTime.h"
#include <esp_task_wdt.h> //Biblioteca do watchdog
//#define MDASH_APP_NAME "BLE_31364"
//#include <mDash.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>


void BLE();
void connect2();
void HoraData1();
void HoraData2();
void Temperaturas();
void conectarEnviar();
void Correntes();
void Tensoes();
void DHCP();

//***********Define o nome da rede, senha para conexão e os endereços para
conexão*******************

const char* rede = "SmartVac Telemetria";
const char* senha = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; procurar
const char* SERVIDOR = "web.smartvac.app";
```

```
int PORTA = 1883;

//************************************Define os itens do MQTT
**********************************

WiFiClient Client;
PubSubClient MQTT(Client);

//***************Definição dos pinos para os sensores de temperatura, corrente e
tensão****************

#define ONE_WIRE_BUS_PINA   14   // Temperatura
#define ONE_WIRE_BUS_PINB   36   // Temperatura

#define PIN_CURR_R             34   // Corrente R
#define PIN_CURR_S             36  // Corrente S
#define PIN_CURR_T             39   // Corrente T

#define PIN_VOLT_R             35   // Tensão R
#define PIN_VOLT_S             32   // Tensão S
#define PIN_VOLT_T             33   // Tensão T


//#define DEVICE_PASSWORD    "oh099QByIVRdERWq4CRijnA"

//******************************Variaveis
Globais*****************************************************

float TempIns;
float TempRet;
float TempSuc;
float TempDes;
float TempExt;
float TempExt2;
float BLERet;
float BLESuc;
float BLEIns;
float BLELinha;
float BLEExt;

float BLEX;
```

```
float BLEY;
float BLEZ;

float Bat_ins;
float Bat_ret;
float Bat_linha;
float Bat_suc;
float Bat_ext;


uint16_t voltage;
int16_t accX;
int16_t accY;
int16_t accZ;




std::map<std::string, float> macTemperatures;

time_t timer;
time_t timeStemp;
int diferenca=0;
int difOitoHoras=0;

//***************************Abre a instância OneWire*************************************

OneWire oneWireA(ONE_WIRE_BUS_PINA);
OneWire oneWireB(ONE_WIRE_BUS_PINB);

//*****************Passagem de dados do one wire para o Dallas*******************************

DallasTemperature sensorsA(&oneWireA);
DallasTemperature sensorsB(&oneWireB);

// Declaração das variaveis para medição de corrente e tensão

EnergyMonitor emon1,emon2,emon3,emon4,emon5,emon6;

double Irms1,Irms2, Irms3;
double Vrms4, Vrms5, Vrms6;
```

```
//****************************Calibração*****************************************************

#define VOLT_CAL1 227.87 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL2 000 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM MULTÍMETRO)
#define VOLT_CAL3 000  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM MULTÍMETRO)
#define CURR_CAL1 15.94   //17.7VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL2 000  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM MULTÍMETRO)
#define CURR_CAL3 000  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM MULTÍMETRO)

//*****************Declaração endereço sensores de temperatura*****************************

DeviceAddress Probe01 = { 0x28, 0x6F, 0x2A, 0x95, 0xF0, 0x01, 0x3C, 0x88  }; //Insuflamento
DeviceAddress Probe02 = { 0x28, 0xAD, 0xD3, 0x56, 0xB5, 0x01, 0x3C, 0x99  }; //Retorno
DeviceAddress Probe03 = { 0x28, 0x7D, 0x6D, 0x95, 0xF0, 0x01, 0x3C, 0x05  }; //Sucção
DeviceAddress Probe04 = { 0x28, 0xC1, 0x30, 0x95, 0xF0, 0x01, 0x3C, 0x06  }; //Descarga
DeviceAddress Probe05 = { 0x28, 0x9C, 0x44, 0x56, 0xB5, 0x01, 0x3C, 0xA5  }; //Externa
DeviceAddress Probe06 = { 0x28, 0xC9, 0x27, 0x95, 0xF0, 0x01, 0x3C, 0x8A  }; //Externa Reserva
(entrada do condensador)

//*****************Setup do hardware*********************************************************


BLEScan* pBLEScan;

/*-------------------------------------------------------------------------------------------
--------------------
// Lista de sensores BLE. SEMPRE SEGUIR A ORDEM: Insuflamento, Retorno, Sucção, Linha de
Líquido/Descarga, Externa.
-------------------------------------------------------------------------------------------
----------------------*/
std::string macAddresses[] = {
    "bc:57:29:0e:19:74", //Insuflamento
    "bc:57:29:0e:26:25", //Retorno
    "bc:57:29:0e:26:38", // Sucção
    "bc:57:29:0e:19:8b", // Linha De Líquido/ Descarga
    "xx:xx:xx:xx:xx:xx" //Externa
};
```

```
// Keep track of the current MAC address index we are looking for
int currentMacIndex = 0;
const int numMacAddresses = sizeof(macAddresses) / sizeof(macAddresses[0]);

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
    void onResult(BLEAdvertisedDevice advertisedDevice) {
        std::string strServiceData = advertisedDevice.getServiceData();
        std::string mac = advertisedDevice.getAddress().toString();



        // Only proceed if the MAC address matches the current one we are looking for
        if(mac == macAddresses[currentMacIndex]){
            uint8_t* payload = (uint8_t*)strServiceData.c_str();
            int len = strServiceData.length();

            Serial.print("Received payload from ");
            Serial.print(mac.c_str());
            Serial.print(": ");
            for(int i = 0; i < len; i++){
                if(payload[i] < 16) Serial.print("0");  // If less than 16, prepend with '0'
to maintain 2 hex digits
                Serial.print(payload[i], HEX);
            }
            Serial.println();

            int offset = 2; // Starting offset after frame type and version tag

            uint8_t sensorMask = payload[offset++];
            if(sensorMask & 0x01) { // Voltage
                uint16_t voltage = (payload[offset] << 8) | payload[offset + 1];
                offset += 2;
                Serial.print("Voltage: ");
                Serial.print(voltage);
                Serial.println(" mV");
                    if (mac == macAddresses[0]) { //Insuflamento

                       Bat_ins = voltage;
                  }
                  else if (mac == macAddresses[1]) { //Retorno
```

```
            Bat_ret = voltage;
        }
          else if (mac == macAddresses[2]) { //Sucção

            Bat_suc = voltage;
        }
          else if (mac == macAddresses[3]) { //Linha de liquido

           Bat_linha = voltage;
          }

          else if (mac == macAddresses[4]) { //Externa

           Bat_ext = voltage;
          }
}

if(sensorMask & 0x02) { // Temperature
    uint16_t tempRaw = (payload[offset] << 8) | payload[offset + 1];
    float temp = tempRaw / 256.0;
    offset += 2;
    Serial.print("Temperature: ");
    Serial.print(temp);
    Serial.println(" °C");
      if (mac == macAddresses[0]) { //Insuflamento
         // ArraySensores[0] = temp;
          BLEIns = temp;
      }
      else if (mac == macAddresses[1]) { //Retorno

          BLERet = temp;
      }
        else if (mac == macAddresses[2]) { //Sucção

          BLESuc = temp;
      }
        else if (mac == macAddresses[3]) { //Linha de liquido

           BLELinha = temp;
          }
```

```
                else if (mac == macAddresses[4]) { //Externa

                 BLEExt = temp;
                }
            }

            if(sensorMask & 0x08) { // Acceleration
                int16_t accX = (payload[offset] << 8) | payload[offset + 1];
                offset += 2;
                int16_t accY = (payload[offset] << 8) | payload[offset + 1];
                offset += 2;
                int16_t accZ = (payload[offset] << 8) | payload[offset + 1];
                offset += 2;
                Serial.print("Acceleration - X: ");
                Serial.print(accX);
                Serial.println(" mg");
                Serial.print("Y: ");
                Serial.print(accY);
                Serial.println(" mg");
                Serial.print("Z: ");
                Serial.print(accZ);
                Serial.println(" mg");

                if (mac == macAddresses[3]) // Se for o sensor de sucção, lê as vibrações
                {

                      BLEX = accX;
                      BLEY = accY;
                      BLEZ = accZ;

                 }
            }

            Serial.println("--------------------"); // Separator for readability

            // Move to the next MAC address in the list
            currentMacIndex = (currentMacIndex + 1) % numMacAddresses;
        }
    }

};
```

```
void setup() {

  // Serial para leitura dos dados

  Serial.begin(115200);

  // Inicia o Wifi
  WiFi.mode(WIFI_STA);
  WiFi.begin(rede, senha);

  //Inicia o password do MDash
  //mDashBegin(DEVICE_PASSWORD);

  // Estabelece o DHCP para conexão com ip dinâmico
  DHCP();

  //Inicia sensores
  sensorsA.begin();
  sensorsB.begin();

  // Seta a resolução do sensor, menor mais rápido
  sensorsA.setResolution(Probe01, 12);
  sensorsA.setResolution(Probe02, 12);
  sensorsB.setResolution(Probe03, 12);
  sensorsB.setResolution(Probe04, 12);
  sensorsA.setResolution(Probe05, 12);
  sensorsB.setResolution(Probe06, 12);

 // Define os pinos e resolução do sensor de corrente
  emon1.current(PIN_CURR_R, CURR_CAL1);       // Current: input pin, calibration. Cur Const=
Ratio/BurdenR. 1800/62 = 29.
  emon2.current(PIN_CURR_S, CURR_CAL2);       // Current: input pin, calibration. Cur Const=
Ratio/BurdenR. 1800/62 = 29.
  emon3.current(PIN_CURR_T, CURR_CAL3);       // Current: input pin, calibration. Cur Const=
Ratio/BurdenR. 1800/62 = 29.

  // Definição do pino para tensão
  emon4.voltage(PIN_VOLT_R, VOLT_CAL1, 1.7); //PASSA PARA A FUNÇÃO OS PARÂMETROS (PINO
ANALÓGIO / VALOR DE CALIBRAÇÃO / MUDANÇA DE FASE)
  emon5.voltage(PIN_VOLT_S, VOLT_CAL2, 1.7); //PASSA PARA A FUNÇÃO OS PARÂMETROS (PINO
ANALÓGIO / VALOR DE CALIBRAÇÃO / MUDANÇA DE FASE)
```

```
  emon6.voltage(PIN_VOLT_T, VOLT_CAL3, 1.7); //PASSA PARA A FUNÇÃO OS PARÂMETROS (PINO ANALÓGIO / VALOR DE CALIBRAÇÃO / MUDANÇA DE FASE)


//*****************************************************************
  //Indica para o objeto "MQTT" em que servidor e em que porta iremos nos conectar

   MQTT.setServer(SERVIDOR,PORTA);
   MQTT.setCallback(mqtt_callback);

   connect2();

   // Inicia o timer
   HoraData1();

   //Watchdog
   esp_task_wdt_init(10800, true);
   esp_task_wdt_add(NULL);

   //BLE
    BLEDevice::init("");
    pBLEScan = BLEDevice::getScan();
    pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
    pBLEScan->setActiveScan(true);
    pBLEScan->setInterval(100);
    pBLEScan->setWindow(99);
  }


//*****************Função loop*****************************************************

void loop() {

  conectarEnviar();

  }


//*****************Função do timer*************************************************

void HoraData1()
```

```
{

   const char tenn[]="CST-3";
   DateTime.setTimeZone(tenn);
   DateTime.setServer("ntp02.oal.ul.pt");
   DateTime.begin();
   int timer = DateTime.getTime();

   timeval tv;//Cria a estrutura temporaria para funcao abaixo.
   tv.tv_sec = timer;//Atribui minha data atual.
   settimeofday(&tv, NULL);//Atualiza a data e hora

   //String hora = String(String(dia) + String("/") + String(mes) + String("/") + String(ano)
+ String("  ") + String(hora) + String(":") + String(minuto) + String(":") + String(segundo));

}

//*****************Função do timer
*****************************************************************************

void HoraData2()
{

   struct tm data;

   time_t tt = time(NULL);//Obtem o tempo atual em segundos. Utilize isso sempre que precisar
obter o tempo atual
   //data = *gmtime(&tt);//Converte o tempo atual e atribui na estrutura

   diferenca=tt-timeStemp;//faz a conta para verificar se a diferença é de 2 segundos para os
envios

   difOitoHoras=tt-timer; //faz a conta para verificar se a diferença é de 28800 segundos para
os envios



   timeStemp=tt;

}
//*****************Função para publicar em formato
JSON*****************************************************
```

```
void Publish() {

JsonDocument doc1;
//  StaticJsonDocument<300> doc;

doc1["t"] = timeStemp;
doc1["s21289"] = Irms1;
//doc1["s91345"] = Irms2;
//doc1["s91346"] = Irms3;
doc1["s21288"] = Vrms4;
//doc1["s91342"] = Vrms5;
//doc1["s91343"] = Vrms6;
doc1["s21290"] = BLEX;


String STD1;

JsonDocument doc2;
doc2["t"] = timeStemp;
doc2["s21284"] = BLEIns;
doc2["s21285"] = BLERet;
doc2["s21286"] = BLESuc;
doc2["s21287"] = BLELinha;
//doc2["BLEext"] = BLEExt;
doc2["s21291"] = BLEY;
doc2["s21292"] = BLEZ;

String STD2;

JsonDocument doc3;
doc3["t"] = timeStemp;
doc3["s21293"] = Bat_ins;
doc3["s21294"] = Bat_ret;
doc3["s21295"] = Bat_suc;
doc3["s21296"] = Bat_linha;
//doc3["bat_ext"] = Bat_ext;

String STD3;

serializeJson(doc1, STD1);
```

```
serializeJson(doc2, STD2);
serializeJson(doc3, STD3);
Serial.println (STD1);
Serial.println (STD2);
Serial.println (STD3);



//*****************MQTT.publish aceita apenas char, as próximas linhas convertem a string em char*****************

    int tamanho = STD1.length() + 1;    //Define o tamanho da String
    char mensa1[tamanho];               //Cria um array de char com o tamanho da String
    STD1.toCharArray(mensa1, tamanho);   //Converte a String  para char e atribui os valores ao array

    int tamanho2 = STD2.length() + 1;    //Define o tamanho da String
    char mensa2[tamanho2];               //Cria um array de char com o tamanho da String
    STD2.toCharArray(mensa2, tamanho2);   //Converte a String  para char e atribui os valores ao array

    int tamanho3 = STD3.length() + 1;    //Define o tamanho da String
    char mensa3[tamanho3];               //Cria um array de char com o tamanho da String
    STD3.toCharArray(mensa3, tamanho3);   //Converte a String  para char e atribui os valores ao array



//*****************************************************************************************************
    MQTT.publish("v4/matr0537",mensa1); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0537",mensa2); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0537",mensa3); // Envio de dados para determinado lugar do tópico
    delay (1000);
    Serial.println("Enviou");

    delay(60000);

   }
```

```
//*****************Função de conexão no
MQTT*************************************************************************


void conectarEnviar()  {


     MQTT.loop();

     if (WiFi.status() == WL_CONNECTED && MQTT.connected()) {

     Serial.println("------------------");
     Serial.println("Wifi conectado e servidor conectados");
     Serial.println("------------------");

     // Calcula a ultima atualização horária e roda as leituras
     if(difOitoHoras > 28600)
     {

      HoraData1();
      Tensoes();
      Temperaturas();
      Correntes();
      BLE();
      Publish();

      difOitoHoras=0;

     }
     else
     {

     HoraData2();

     if(diferenca>2)
     {

     Tensoes();
     Temperaturas();
```

```
      Correntes();
   BLE();
      Publish();

      }
      }

      }
      else if( WiFi.status() != WL_CONNECTED || !MQTT.connected())
      {
      Serial.println("------------------");
      Serial.println("Wifi ou servidor desconectado");
      Serial.println("------------------");

      DHCP();
      connect2();

      }
  }


//*****************Função de leitura das
temperaturas*********************************************************

void Temperaturas() {

  sensorsA.requestTemperatures();
  sensorsB.requestTemperatures();

 // Serial.println("######TEMPERATURAS######");

  TempIns = (sensorsA.getTempC(Probe01));
  //Serial.print("Insuflamento: ");
  //Serial.print(TempIns);
  //Serial.println("°C");

  TempRet = (sensorsA.getTempC(Probe02));
  //Serial.print("Retorno: ");
  //Serial.print(TempRet);
  //Serial.println("°C");
```

```
  TempSuc = (sensorsB.getTempC(Probe03));
  //Serial.print("Sucção: ");
  //Serial.print(TempSuc);
  //Serial.println("°C");

  TempDes = (sensorsB.getTempC(Probe04));
  //Serial.print("Descarga: ");
  //Serial.print(TempDes);
  //Serial.println("°C");

  TempExt = (sensorsA.getTempC(Probe05));
  //Serial.print("Externa: ");
  //Serial.print(TempExt);
  //Serial.println("°C");

  TempExt2 = (sensorsB.getTempC(Probe06));
  //Serial.print("Retorno Condensador: ");
  //Serial.print(TempExt2);
  //Serial.println("°C");

}

//*****************Função de leitura das
correntes*****************************************************

void Correntes() {

  Irms1 = emon1.calcIrms(1996);//Para 50Hz 1480 e para 60Hz 1996
  Irms2 = emon2.calcIrms(1996);//Para 50Hz 1480 e para 60Hz 1996
  Irms3 = emon3.calcIrms(1996);//Para 50Hz 1480 e para 60Hz 1996

}

//*****************Função de leitura das
tensões*******************************************************

void Tensoes()
{

emon4.calcVI(17,1000); //FUNÇÃO DE CÁLCULO (17 SEMICICLOS, TEMPO LIMITE PARA FAZER A MEDIÇÃO)
  Vrms4 = emon4.Vrms;     //VARIÁVEL RECEBE O VALOR DE TENSÃO RMS OBTIDO
```

```
  emon5.calcVI(17,1000); //FUNÇÃO DE CÁLCULO (17 SEMICICLOS, TEMPO LIMITE PARA FAZER A
MEDIÇÃO)
  Vrms5 = emon5.Vrms;     //VARIÁVEL RECEBE O VALOR DE TENSÃO RMS OBTIDO

  emon6.calcVI(17,1000); //FUNÇÃO DE CÁLCULO (17 SEMICICLOS, TEMPO LIMITE PARA FAZER A
MEDIÇÃO)
  Vrms6 = emon6.Vrms;     //VARIÁVEL RECEBE O VALOR DE TENSÃO RMS OBTIDO

}

//**********************Função de leitura
BLE************************************************************

void BLE() {

  // Início da varredura BLE
  BLEScanResults foundDevices = pBLEScan->start(5, false);

  // Limpa os resultados da varredura BLE
  pBLEScan->clearResults();
}




//*****************Função de login e conexão
MQTT*****************************************************

void connect2() {

int cont=0;

    while (!MQTT.connect("9906", "matr4", "canudos92sc"))
    {
        Serial.println("* Tentando se conectar ao Broker MQTT: ");
        if (MQTT.connect("9906", "matr4", "canudos92sc"))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe("/v4/matr0537");
        }
```

```
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentativa de conexao em 1s");
            delay(1000);

            cont++;

            if(WiFi.status() != WL_CONNECTED && cont<10)
            {
            DHCP();
            }
            else
            {
            cont=0;
            Serial.println("Teste");
            }
        }
    }
}

//*****************Função de callback do
servidor*********************************************************

void messageReceived(String &topic, String &payload) {
  Serial.println("incoming: " + topic + " - " + payload);      // Lê o que o servidor envia

  MQTT.setCallback(mqtt_callback);
}


void mqtt_callback(char* topic, byte* payload, unsigned int length)
{
    String msg;

    //obtem a string do payload recebido
    for(int i = 0; i < length; i++)
    {
       char c = (char)payload[i];
       msg += c;
    }
```

```
    Serial.print("[MQTT] Mensagem recebida: ");
    Serial.println(msg);
}



//*****************Função do DHCP IP
Dinâmico*****************************************************

void DHCP()
{

  if(WiFi.status() != WL_CONNECTED) {
    Serial.println("Reconectando no wifi...");
    WiFi.disconnect();
    WiFi.reconnect();
    delay(500);
  }
  else if(WiFi.status() == WL_CONNECTED)
  {
    return;
  }
}



//***************************************
```

Revision #1
Created 9 October 2024 12:40:24 by Patrick Leal
Updated 9 October 2024 12:40:53 by Patrick Leal