Braskem RS

- 9425 ckt 1
- 9425 ckt 2
- 9229 ckt 2
- 12842
- 9945
- 9312
- 13318
- 9229 ckt 1
- 17407
- 9147 ckt 1
- 9147 ckt 2
- 9312
- 9433
- 9495
- 9496
- 9502
- 9503
- 9504
- 9603
- 9604
- 9606
- 9607
- 9692
- 9758
- 9905
- 9906
- 9955 CKT 1

- 9955 CKT 2
- 10026
- 10108
- 10254
- 12837
- 12838
- 12842
- 12845
- 13132 ckt 2
- 13318
- 17406
- 17407
- 23916
- 36512
- 36514
- 36627
- 9575
- 9824 ckt 2
- 9824 ckt 1
- 9571
- 10246
- 10243
- 12388
- 9314
- 10245
- 36516
- 10244
- 9825 ckt 2
- 9428
- 9825 ckt 1
- 9428 ckt 1
- 36513
- 17336
- 36515

- 9736 ckt 1
- 9441 ckt 2
- 9441 ckt 1
- 9443 ckt 1
- 9736 ckt 2
- 9439 ckt 1
- 9999
- 9998
- 9994
- 9955
- 9955 ckt 2
- 9945
- 9906
- 9905
- 9758
- 9330
- 10139 CKT 2
- 10139 CKT 1
- 9904
- 9338
- 9436
- 9494
- 9497
- 9498
- 9978
- 9980
- 9980 CKT 2
- 9996
- 9605
- 9321
- 10052 CKT 1
- 10052 CKT 2
- 12826
- 12828

- 12862
- 13217
- 33213
- 36517
- 36518
- 10017

9425 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

 /*********************** ### Circuito 1 Chiller 322 Braskem RS ### *********/
/********************** ### Usar versão de código com  6 sensores BLE que está no GitHub ###
**************/
/********************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********* ### Se atentar a página "initserver.cpp" e verificar se não tem nenhum envio de
pacotes comentado ### ************/

#define NETWORK_CLIENT "SmartVac Telemetria"//nome da rede wifi do cliente

#define PASSW  "procurar_no_gerenciador" //senha do wifi do cliente procurar no bitwarden

#define VOLT_CAL  180.45  //calibracao do sensor de tensao

#define VOLT_CAL_S  179.24  //calibracao do sensor de tensao

#define VOLT_CAL_TT  186.05  //calibracao do sensor de tensao

#define CURRENT_CAL  14.4 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14.3  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.5  //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9425"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/SC0004"  //topico de envio, fornecido pelo spo  EQUIPAMENT_TOPIC

#define INS_SENSOR "s201"  //endereco sensor de insuflamento

#define RET_SENSOR "s202"  //endereco sensor de retorno

#define SUC_SENSOR "s212"  //endereco sensor de sucção

#define LL_SENSOR "s213"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s20850"  //endereco sensor de externa

#define SAD_CONDES "s20849"  //endereco sensor de externa

#define VOLT_SENSOR_R "s214" //endereco sensor de tensao

#define VOLT_SENSOR_S "s215" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s216" //endereco sensor de tensao

#define CURR_SENSOR_R "s203" //endereco sensor de corrente

#define CURR_SENSOR_S "s204" //endereco sensor de corrente

#define CURR_SENSOR_TT "s205" //endereco sensor de corrente

#define BAT_SENSOR_INS "s20841" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20842" //endereco sensor de corrente
```

```
#define BAT_SENSOR_SUC "s20843" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20844" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s217" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s218" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s219" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:19:48" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:25:db"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:5a" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:ed"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:25:df"  //endereço para a leitura do sensor de
entrada da condensação

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:19:c7"  //endereço para a leitura do sensor de
saida da condensação

#endif
```

9425 ckt 2

```
/********************** ### Circuito 2 Chiller 9425 Braskem RS ### *********/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********* ### Se atentar a página "initserver.cpp" e verificar se não tem nenhum envio de
pacotes comentado ### ************/
/******** ### Esse circuito NÃO ENVIA TENSÃO, somente o circuito 1 envia tensão ###
*************/
/******** ### Esse circuito ENVIA INSU E RET, pois o chiller é de dois circuitos ###
*************/
/******** ### Esse circuito NÃO ENVIA AS BATERIAS DOS SENSORES BLE, somente o circuito 1 envia
esses dados ### *************/


#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria" //nome da rede

#define PASSW  "procurar_no_gerenciador" //senha

#define VOLT_CAL  454  //calibracao do sensor de tensao

#define VOLT_CAL_S  82  //calibracao do sensor de tensao

#define VOLT_CAL_TT  119.5  //calibracao do sensor de tensao

#define CURRENT_CAL  18.04 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.3  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  16.04 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente
```

```
#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9425b"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/SC0004b"  //topico de envio, fornecido pelo spo  EQUIPAMENT_TOPIC

#define INS_SENSOR "s20845"  //endereco sensor de saida  ckt 2

#define RET_SENSOR "s20846"  //endereco sensor de saida ckt 2

#define SUC_SENSOR "s20848"  //endereco sensor de sucção   2

#define LL_SENSOR "s20847"   //endereco sensor de descarga 2

#define ENT_CONDES "s20852"  //endereco sensor entrada cond 2

#define SAD_CONDES "s20851"  //endereco sensor saida cond 2

#define VOLT_SENSOR_R "s214" //endereco sensor de tensao

#define VOLT_SENSOR_S "s215" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s216" //endereco sensor de tensao

#define CURR_SENSOR_R "s206" //endereco sensor de corrente

#define CURR_SENSOR_S "s207" //endereco sensor de corrente

#define CURR_SENSOR_TT "s208" //endereco sensor de corrente

#define BAT_SENSOR_INS "s20841" //endereco sensor de corrente
```

```
#define BAT_SENSOR_RET "s20842" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20843" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20844" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s220" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s221" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s222" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:25:f1" //entrada de água 2

#define SENS_TEMP_INSU "bc:57:29:0e:19:db"  // saída de água 2

#define SENS_TEMP_LL "bc:57:29:0e:26:23" //descarga ckt 2

#define SENS_TEMP_SUC "bc:57:29:0e:19:85"  //sucção ckt 2

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:51"  //endereço para a leitura do sensor de
entrada da condensação

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:19:97"  //endereço para a leitura do sensor de
saida da condensação

#endif
```

9229 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

/********************** ### Circuito 2 Chiller CPD Braskem RS ### *********/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********* ### Se atentar a página "initserver.cpp" e verificar se não tem nenhum envio de pacotes comentado ### ************/
/******** ### Esse circuito NÃO ENVIA TENSÃO, somente o circuito 1 envia tensão ### *************/
/******** ### Esse circuito NÃO ENVIA INSU E RET, somente o circuito 1 envia esses dados ### *************/
/******** ### Esse circuito NÃO ENVIA AS BATERIAS DOS SENSORES BLE, somente o circuito 1 envia esses dados ### *************/

#define NETWORK_CLIENT "SmartVac Telemetria"//nome da rede wifi do cliente

#define PASSW  "procurar_no_gerenciador"//senha do wifi di cliente

#define VOLT_CAL  181.04 //calibracao do sensor de tensao

#define VOLT_CAL_S  179.1 //calibracao do sensor de tensao

#define VOLT_CAL_TT  183.7 //calibracao do sensor de tensao

#define CURRENT_CAL  14.8 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14.79  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.22 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente
```

```
#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9229b"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/SC0002b"  //topico de envio, fornecido pelo spo  EQUIPAMENT_TOPIC

//#define INS_SENSOR "s20924"  //endereco sensor de saida

//#define RET_SENSOR "s20925"  //endereco sensor de retorno

#define SUC_SENSOR "s20930"  //endereco sensor de sucção

#define LL_SENSOR "s20931"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20926" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20932" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20933" //endereco sensor de tensao

#define CURR_SENSOR_R "s20946" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20947" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20948" //endereco sensor de corrente ckt 1

//#define BAT_SENSOR_INS "s20942" //endereco sensor de corrente

//#define BAT_SENSOR_RET "s20943" //endereco sensor de corrente

//#define BAT_SENSOR_SUC "s20944" //endereco sensor de corrente
```

```
//#define BAT_SENSOR_LL "s20945" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20939" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20940" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20941" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:19:56" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:19:7c"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:26:17" //descarga ckt 2

#define SENS_TEMP_SUC "bc:57:29:0e:19:a6"  //sucção ckt 2

#endif
```

12842

```
/********************** ### 12842/AC-40-4 Braskem RS ### *********/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********* ### Se atentar a página "initserver.cpp" e verificar se não tem nenhum envio de
pacotes comentado ### ************/

#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"//nome da rede wifi do cliente

#define PASSW  "procurar_no_gerenciador"// network password

#define VOLT_CAL  113.42 //calibracao do sensor de tensao

#define VOLT_CAL_S  108.7 //calibracao do sensor de tensao

#define VOLT_CAL_TT  506.14 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  13.125 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "12842"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0517"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20949"  //endereco sensor de saida

#define RET_SENSOR "s20950"  //endereco sensor de retorno

#define SUC_SENSOR "s20951"  //endereco sensor de sucção

#define LL_SENSOR "s20952"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20953" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20962" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20963" //endereco sensor de tensao

#define CURR_SENSOR_R "s20954" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20964" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20965" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s20958" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20959" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20960" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20961" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20955" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20956" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Z_SUC "s20957" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 2b" //entrada de água

#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: 75"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 26: 00" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 88"  //sucção ckt 1

#endif
```

9945

```
/**************************##############################*******************************/
/*********************** ### Self 9945 Braskem RS ### *********************************/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********************#########################################################*******************/

#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"//nome da rede wifi

#define PASSW  "procurar_no_gerenciador"//senha da rede

#define VOLT_CAL  147.79 //calibracao do sensor de tensao

#define VOLT_CAL_S  193.9 //calibracao do sensor de tensao

#define VOLT_CAL_TT  192.2 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9945"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0522"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21029"  //endereco sensor de saida

#define RET_SENSOR "s21030"  //endereco sensor de retorno

#define SUC_SENSOR "s21031"  //endereco sensor de sucção

#define LL_SENSOR "s21032"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21033" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21042" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21043" //endereco sensor de tensao

#define CURR_SENSOR_R "s21034" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21044" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21045" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21038" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21039" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21040" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21041" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21035" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21036" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Z_SUC "s21037" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 25: f9" //entrada de água

#define SENS_TEMP_INSU "bc: 57: 29: 0e: 2e: fe"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: ac" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 4c"  //sucção ckt 1

#endif
```

9312

```
/**************************##############################*******************************/
/*********************** ### Self 9312 Braskem RS ### *********************************/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********************##########################################################*********/


#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"//rede wifi

#define PASSW  "procurar_no_gerenciador"//senha da rede

#define VOLT_CAL  212 //calibracao do sensor de tensao

#define VOLT_CAL_S  94 //calibracao do sensor de tensao

#define VOLT_CAL_TT  208.39 //calibracao do sensor de tensao

#define CURRENT_CAL  14.28 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14.28  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.28 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39    //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9312"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0521"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21004"  //endereco sensor de saida

#define RET_SENSOR "s21005"  //endereco sensor de retorno

#define SUC_SENSOR "s21008"  //endereco sensor de sucção

#define LL_SENSOR "s21009"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21006" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21012" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21013" //endereco sensor de tensao

#define CURR_SENSOR_R "s21007" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21014" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21015" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21022" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21023" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21024" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21025" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21016" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s21017" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21018" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 26: 35" //entrada de água

#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: e0"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 5b" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 2c"  //sucção ckt 1

#endif
```

13318

```
/**************************##############################*******************************/
/*********************** ### Self 13318 Braskem RS ### **********************************/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/****************################## Monofásico ##################************************/
/******************** ### Scomentar os envios em iniserver.cpp conforme os sensores na
presente página. ### ********************/
/********************############################################*******************/

#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"//rede wifi

#define PASSW  "procurar_no_gerenciador"//senha

#define VOLT_CAL  181.62  //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  24.27 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard
```

```
#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "13318"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0520"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20991"  //endereco sensor de saida

#define RET_SENSOR "s20992"  //endereco sensor de retorno

#define SUC_SENSOR "s20993"  //endereco sensor de sucção

#define LL_SENSOR "s20994"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20995" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s20996" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21000" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21001" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21002" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21003" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_X_SUC "s20997" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20998" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20999" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:19:9e" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:f6"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:b3" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:37"  //sucção ckt 1

#endif
```

9229 ckt 1

```
/********************** ### Circuito 1 Chiller 9229 Braskem RS ### *********/
/******************** ### Se atentar ás credenciais da rede Wi-Fi ### ********************/
/********* ### Se atentar a página "initserver.cpp" e verificar se não tem nenhum envio de pacotes comentado ### ************/


#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"//rede

#define PASSW  "procurar_no_gerenciador"//senha

#define VOLT_CAL  182.78  //calibracao do sensor de tensao

#define VOLT_CAL_S  184.76 //calibracao do sensor de tensao

#define VOLT_CAL_TT  183.7 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9229"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/SC0002"  //topico de envio, fornecido pelo spo  EQUIPAMENT_TOPIC

#define INS_SENSOR "s20924"  //endereco sensor de saida

#define RET_SENSOR "s20925"  //endereco sensor de retorno

#define SUC_SENSOR "s20928"  //endereco sensor de sucção

#define LL_SENSOR "s20929"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20926" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20932" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20933" //endereco sensor de tensao

#define CURR_SENSOR_R "s20927" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20934" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20935" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s20942" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20943" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20944" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20945" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20936" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20937" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Z_SUC "s20938" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 56" //entrada de água

#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: 7c"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 55" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 9a"  //sucção ckt 1

#endif
```

17407

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"// rede

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxx"//senha da rede procurar no gerenciador

#define VOLT_CAL  520.88  //calibracao do sensor de tensao

#define VOLT_CAL_S  452.75  //calibracao do sensor de tensao

#define VOLT_CAL_TT  462.42  //calibracao do sensor de tensao

#define CURRENT_CAL  4.98 //calibracao do sensor de corrente

#define CURRENT_CAL_S 4.88  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  4.50  //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "17407"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0516"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20890"  //endereco sensor de saida

#define RET_SENSOR "s20891"  //endereco sensor de retorno

#define SUC_SENSOR "s20892"  //endereco sensor de sucção

#define LL_SENSOR "s20893"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20894" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20895" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20896" //endereco sensor de tensao

#define CURR_SENSOR_R "s20897" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20898" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20899" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s20903" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20904" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20905" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20906" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20900" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20901" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20902" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 8c" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:2f:17"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:d9" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:eb"  //sucção ckt 1

#endif
```

9147 ckt 1

```
#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 = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; procurar no
gerenciador
```

```
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             39   // Corrente R
#define PIN_CURR_S             33  // Corrente S
#define PIN_CURR_T             35   // Corrente T

#define PIN_VOLT_R             32   // Tensão R
#define PIN_VOLT_S             34   // Tensão S
#define PIN_VOLT_T             36   // 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 81.95 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL2 81.89 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL3 76.80 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL1 16.55  //17.7VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL2 16.55  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL3 19.04  //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:2e:d8", //Insuflamento
    "bc:57:29:0e:25:e4", //Retorno
    "bc:57:29:0e:19:3d", // Sucção
    "bc:57:29:0e:19:ce", // 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["s21313"] = Irms1;
doc1["s21320"] = Irms2;
doc1["s21321"] = Irms3;
doc1["s21312"] = Vrms4;
doc1["s21318"] = Vrms5;
doc1["s21319"] = Vrms6;
doc1["s21322"] = BLEX;



String STD1;

JsonDocument doc2;
doc2["t"] = timeStemp;
doc2["s21310"] = BLEIns;
doc2["s21311"] = BLERet;
doc2["s21314"] = BLESuc;
doc2["s21315"] = BLELinha;
//doc2["BLEext"] = BLEExt;
doc2["s21323"] = BLEY;
```

```
doc2["s21324"] = BLEZ;

String STD2;

JsonDocument doc3;
doc3["t"] = timeStemp;
doc3["s21328"] = Bat_ins;
doc3["s21329"] = Bat_ret;
doc3["s21330"] = Bat_suc;
doc3["s21331"] = 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/matr0539",mensa1); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0539",mensa2); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0539",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("9147", "matr4", "canudos92sc"))
    {
        Serial.println("* Tentando se conectar ao Broker MQTT: ");
        if (MQTT.connect("9147", "matr4", "canudos92sc"))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe("/v4/matr0539");
        }
        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;
  }
}



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

9147 ckt 2

```
#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 = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";procurar no
gerenciador
```

```
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             39   // Corrente R
#define PIN_CURR_S             33  // Corrente S
#define PIN_CURR_T             35   // Corrente T

#define PIN_VOLT_R             32   // Tensão R
#define PIN_VOLT_S             34   // Tensão S
#define PIN_VOLT_T             36   // 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 81.95 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL2 81.89 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL3 76.80 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL1 17  //17.7VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL2 17.07  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL3 16.53  //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:2f:16", //Insuflamento
    "bc:57:29:0e:19:c4", //Retorno
    "bc:57:29:0e:19:b9", // Sucção
    "bc:57:29:0e:19:3e", // 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["s21332"] = Irms1;
doc1["s21333"] = Irms2;
doc1["s21334"] = Irms3;
//doc1["s21312"] = Vrms4;
//doc1["s21318"] = Vrms5;
//doc1["s21319"] = Vrms6;
doc1["s21325"] = BLEX;



String STD1;

JsonDocument doc2;
doc2["t"] = timeStemp;
//doc2["s91335"] = BLEIns;
//doc2["s91336"] = BLERet;
doc2["s21316"] = BLESuc;
doc2["s21317"] = BLELinha;
//doc2["BLEext"] = BLEExt;
```

```
doc2["s21326"] = BLEY;
doc2["s21327"] = BLEZ;

String STD2;

JsonDocument doc3;
doc3["t"] = timeStemp;
doc3["s21328"] = Bat_ins;
doc3["s21329"] = Bat_ret;
doc3["s21330"] = Bat_suc;
doc3["s21331"] = 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/matr0539b",mensa1); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0539b",mensa2); // Envio de dados para determinado lugar do tópico
    delay (1000);
    //MQTT.publish("v4/matr0539",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("9147", "matr4", "canudos92sc"))
    {
        Serial.println("* Tentando se conectar ao Broker MQTT: ");
        if (MQTT.connect("9147", "matr4", "canudos92sc"))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe("/v4/matr0539b");
        }
        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;
  }
}


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

9312

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxx"  procurar no gerenciador

#define VOLT_CAL  212 //calibracao do sensor de tensao

#define VOLT_CAL_S  94 //calibracao do sensor de tensao

#define VOLT_CAL_TT  208.39 //calibracao do sensor de tensao

#define CURRENT_CAL  14.28 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14.28  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.28 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39    //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9312"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0521"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21004"  //endereco sensor de saida

#define RET_SENSOR "s21005"  //endereco sensor de retorno

#define SUC_SENSOR "s21008"  //endereco sensor de sucção

#define LL_SENSOR "s21009"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21006" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21012" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21013" //endereco sensor de tensao

#define CURR_SENSOR_R "s21007" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21014" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21015" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21022" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21023" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21024" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21025" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21016" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21017" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21018" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 26: 35" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:19:e0"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:5b" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:2c"  //sucção ckt 1

#endif
```

9433

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  216.94 //calibracao do sensor de tensao

#define VOLT_CAL_S  180 //calibracao do sensor de tensao

#define VOLT_CAL_TT  180 //calibracao do sensor de tensao

#define CURRENT_CAL  12.22 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9433"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0581"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91569"  //endereco sensor de insuflamento

#define RET_SENSOR "s91570"  //endereco sensor de retorno

#define SUC_SENSOR "s91571"  //endereco sensor de sucção

#define LL_SENSOR "s91572"   //endereco sensor de linha de liquido

#define ENT_CONDES "s91582"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91573" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91574" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91578" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91579" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91580" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91581" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91575" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s91576" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Z_SUC "s91577" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:2e:c7" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2f:94"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:26:36" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:5f"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:2e:e7"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9495

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  223.54 //calibracao do sensor de tensao

#define VOLT_CAL_S  180 //calibracao do sensor de tensao

#define VOLT_CAL_TT  180 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9495"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0585"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91625"  //endereco sensor de insuflamento

#define RET_SENSOR "s91626"  //endereco sensor de retorno

#define SUC_SENSOR "s91627"  //endereco sensor de sucção

#define LL_SENSOR "s91628"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91638"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91629" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91630" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91634" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91635" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91636" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91637" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91631" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91632" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91633" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:2e:b9" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:19:47"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:83" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:a8"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:c2"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9496

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  450 //calibracao do sensor de tensao

#define VOLT_CAL_S  180 //calibracao do sensor de tensao

#define VOLT_CAL_TT  180 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9496"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0584"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91611"  //endereco sensor de insuflamento

#define RET_SENSOR "s91612"  //endereco sensor de retorno

#define SUC_SENSOR "s91613"  //endereco sensor de sucção

#define LL_SENSOR "s91614"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91624"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91615" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91616" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91620" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91621" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91622" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91623" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91617" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91618" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91619" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:26:0e" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:fb"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:4d" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:cb"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:c2"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9502

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  111.73 //calibracao do sensor de tensao

#define VOLT_CAL_S  109.58 //calibracao do sensor de tensao

#define VOLT_CAL_TT  37.71 //calibracao do sensor de tensao

#define CURRENT_CAL  11.5 //calibracao do sensor de corrente

#define CURRENT_CAL_S 11.02  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  11.44 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9502"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0548"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91507"  //endereco sensor de insuflamento

#define RET_SENSOR "s91508"  //endereco sensor de retorno

#define SUC_SENSOR "s91509"  //endereco sensor de sucção

#define LL_SENSOR "s91510"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91524"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91511" //endereco sensor de tensao

#define VOLT_SENSOR_S "s91512" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s91513" //endereco sensor de tensao

#define CURR_SENSOR_R "s91514" //endereco sensor de corrente

#define CURR_SENSOR_S "s91515" //endereco sensor de corrente

#define CURR_SENSOR_TT "s91516" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91520" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91521" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91522" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91523" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91517" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91518" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91519" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:26:06" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:e4"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:ab" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:25:dc"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:2e:b3"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9503

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxsxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  109.57 //calibracao do sensor de tensao

#define VOLT_CAL_S  112.13 //calibracao do sensor de tensao

#define VOLT_CAL_TT  34.18 //calibracao do sensor de tensao

#define CURRENT_CAL  16.65 //calibracao do sensor de corrente

#define CURRENT_CAL_S 17.57  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  17.69 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9503"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0549"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91472"  //endereco sensor de insuflamento

#define RET_SENSOR "s91473"  //endereco sensor de retorno

#define SUC_SENSOR "s91474"  //endereco sensor de sucção

#define LL_SENSOR "s91475"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91525"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91476" //endereco sensor de tensao

#define VOLT_SENSOR_S "s91477" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s91478" //endereco sensor de tensao

#define CURR_SENSOR_R "s91479" //endereco sensor de corrente

#define CURR_SENSOR_S "s91480" //endereco sensor de corrente

#define CURR_SENSOR_TT "s91481" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91485" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91486" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91487" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91488" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91482" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91483" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91484" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:26:08" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:19:58"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:26:13" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:94"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:2e:b3"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9504

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  106.63 //calibracao do sensor de tensao

#define VOLT_CAL_S  107.23 //calibracao do sensor de tensao

#define VOLT_CAL_TT  35.52 //calibracao do sensor de tensao

#define CURRENT_CAL  12.17 //calibracao do sensor de corrente

#define CURRENT_CAL_S 12.41  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  11.84 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9504"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0547"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91489"  //endereco sensor de insuflamento

#define RET_SENSOR "s91490"  //endereco sensor de retorno

#define SUC_SENSOR "s91491"  //endereco sensor de sucção

#define LL_SENSOR "s91492"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91506"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91493" //endereco sensor de tensao

#define VOLT_SENSOR_S "s91494" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s91495" //endereco sensor de tensao

#define CURR_SENSOR_R "s91496" //endereco sensor de corrente

#define CURR_SENSOR_S "s91497" //endereco sensor de corrente

#define CURR_SENSOR_TT "s91498" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91502" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91503" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91504" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91505" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91499" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91500" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91501" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:2e:db" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:e0"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:cf" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:76"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:2e:b3"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9603

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  165.76 //calibracao do sensor de tensao

#define VOLT_CAL_S  104.57 //calibracao do sensor de tensao

#define VOLT_CAL_TT  311.8 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14  //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9603"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0528"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21127"  //endereco sensor de saida

#define RET_SENSOR "s21128"  //endereco sensor de retorno

#define SUC_SENSOR "s21129"  //endereco sensor de sucção

#define LL_SENSOR "s21130"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21131" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21140" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21141" //endereco sensor de tensao

#define CURR_SENSOR_R "s21132" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21142" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21143" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21136" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21137" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21138" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21139" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21133" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21134" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21135" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:25:f8"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:40" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:c5"  //sucção ckt 1

#endif
```

9604

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  305.9  //calibracao do sensor de tensao

#define VOLT_CAL_S  98.61  //calibracao do sensor de tensao

#define VOLT_CAL_TT  153.57  //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  13  //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9604"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0527"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21110"  //endereco sensor de saida

#define RET_SENSOR "s21111"  //endereco sensor de retorno

#define SUC_SENSOR "s21112"  //endereco sensor de sucção

#define LL_SENSOR "s21113"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21114" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21123" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21124" //endereco sensor de tensao

#define CURR_SENSOR_R "s21115" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21125" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21126" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21119" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21120" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21121" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21122" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21116" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21117" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21118" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:26:18"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:cd" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:7f"  //sucção ckt 1

#endif
```

9606

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  83.22 //calibracao do sensor de tensao

#define VOLT_CAL_S  82.4 //calibracao do sensor de tensao

#define VOLT_CAL_TT  75.85 //calibracao do sensor de tensao

#define CURRENT_CAL  17.72 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.18  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  15.7 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9606"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0530"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21161"  //endereco sensor de saida

#define RET_SENSOR "s21162"  //endereco sensor de retorno

#define SUC_SENSOR "s21163"  //endereco sensor de sucção

#define LL_SENSOR "s21164"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21165" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21174" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21175" //endereco sensor de tensao

#define CURR_SENSOR_R "s21166" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21176" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21177" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21170" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21171" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21172" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21173" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21167" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21168" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21169" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 2e: d1"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: b8" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 3f"  //sucção ckt 1

#endif
```

9607

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

#define VOLT_CAL  112.57  //calibracao do sensor de tensao

#define VOLT_CAL_S  112.24  //calibracao do sensor de tensao

#define VOLT_CAL_TT  531  //calibracao do sensor de tensao

#define CURRENT_CAL  13.31 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.08  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  10.9  //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9607"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0524"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21059"  //endereco sensor de saida

#define RET_SENSOR "s21060"  //endereco sensor de retorno

#define SUC_SENSOR "s21061"  //endereco sensor de sucção

#define LL_SENSOR "s21062"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21063" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21072" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21073" //endereco sensor de tensao

#define CURR_SENSOR_R "s21064" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21074" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21075" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21068" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21069" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21070" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21071" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21065" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21066" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21067" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:19:b2"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:49" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:bb"  //sucção ckt 1

#endif
```

9692

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar no gerenciador

#define VOLT_CAL  210.38 //calibracao do sensor de tensao

#define VOLT_CAL_S  180 //calibracao do sensor de tensao

#define VOLT_CAL_TT  180 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9692"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0546"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91459"  //endereco sensor de insuflamento

#define RET_SENSOR "s91460"  //endereco sensor de retorno

#define SUC_SENSOR "s91461"  //endereco sensor de sucção

#define LL_SENSOR "s91462"   //endereco sensor de linha de liquido

#define ENT_CONDES "s91540"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91463" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91464" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91468" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91469" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91470" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91471" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91465" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s91466" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Z_SUC "s91467" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:26:32" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:ed"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:26:15" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:ad"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:7d"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

9758

```
#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 = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; 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 263.38 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL2 0000 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM MULTÍMETRO)
#define VOLT_CAL3 0000  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL1 17.88   //17.7VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL2 00000 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL3 0000 //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:2e:c1", //Insuflamento
    "bc:57:29:0e:19:9d", //Retorno
```

```
    "bc:57:29:0e:25:e9", // Sucção
    "bc:57:29:0e:26:10", // Linha De Líquido/ Descarga
    "bc:57:29:0e:26:2b" //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["s91372"] = Irms1;
//doc1["s91345"] = Irms2;
//doc1["s91346"] = Irms3;
doc1["s91371"] = Vrms4;
//doc1["s91342"] = Vrms5;
//doc1["s91343"] = Vrms6;
doc1["s91373"] = BLEX;



String STD1;

JsonDocument doc2;
doc2["t"] = timeStemp;
doc2["s91367"] = BLEIns;
doc2["s91368"] = BLERet;
doc2["s91369"] = BLESuc;
doc2["s91370"] = BLELinha;
doc2["s91380"] = BLEExt;
doc2["s91374"] = BLEY;
doc2["s91375"] = BLEZ;

String STD2;
```

```
JsonDocument doc3;
doc3["t"] = timeStemp;
doc3["s91376"] = Bat_ins;
doc3["s91377c"] = Bat_ret;
doc3["s91378"] = Bat_suc;
doc3["s91379"] = 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/matr0541",mensa1); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0541",mensa2); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0541",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("9758", "matr4", "canudos92sc"))
    {
        Serial.println("* Tentando se conectar ao Broker MQTT: ");
        if (MQTT.connect("9758", "matr4", "canudos92sc"))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe("/v4/matr0541");
        }
        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;
  }
}
```

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

9905

```
//*************************************************************************************************
*
//************* Equipamento monofásico
*******************************************************
//*************************************************************************************************
*****

#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  253 //calibracao do sensor de tensao

#define VOLT_CAL_S  0 //calibracao do sensor de tensao

#define VOLT_CAL_TT  0 //calibracao do sensor de tensao

#define CURRENT_CAL  16.7 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  0 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE
```

```
#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9905"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0583"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91597"  //endereco sensor de insuflamento

#define RET_SENSOR "s91598"  //endereco sensor de retorno

#define SUC_SENSOR "s91599"  //endereco sensor de sucção

#define LL_SENSOR "s91600"   //endereco sensor de linha de liquido

#define ENT_CONDES "s91610"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91601" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91602" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91606" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91607" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91608" //endereco sensor de corrente
```

```
#define BAT_SENSOR_LL "s91609" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91603" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s91604" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91605" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:2e:d6" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:de"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:43" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:3b"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:8a"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

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 = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"; 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;
  }
}
```

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

9955 CKT 1

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX" PROCURAR

#define VOLT_CAL  182.78  //calibracao do sensor de tensao

#define VOLT_CAL_S  184.76 //calibracao do sensor de tensao

#define VOLT_CAL_TT  183.7 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9955"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0535a"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21234"  //endereco sensor de saida

#define RET_SENSOR "s21235"  //endereco sensor de retorno

#define SUC_SENSOR "s21238"  //endereco sensor de sucção

#define LL_SENSOR "s21239"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21236" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21242" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21243" //endereco sensor de tensao

#define CURR_SENSOR_R "s21237" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21244" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21245" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21252" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21253" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21254" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21255" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21246" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21247" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21248" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 56" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: 7c"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 55" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 9a"  //sucção ckt 1

#endif
```

9955 CKT 2

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "XXXXXXXXXXXXXXXXXXXXXXXXXX" PROCURAR

#define VOLT_CAL  71.47 //calibracao do sensor de tensao

#define VOLT_CAL_S  72.23 //calibracao do sensor de tensao

#define VOLT_CAL_TT  79.51 //calibracao do sensor de tensao

#define CURRENT_CAL  18.11 //calibracao do sensor de corrente

#define CURRENT_CAL_S 17.29  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  16.09 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "9955"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0535"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "xxxxxx"  //endereco sensor de saida

#define RET_SENSOR "xxxxxxx"  //endereco sensor de retorno

#define SUC_SENSOR "s21240"  //endereco sensor de sucção

#define LL_SENSOR "s21241"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s21256" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21257" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21258" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_RET "xxxxxx" //endereco sensor de corrente

#define BAT_SENSOR_SUC "xxxxxx" //endereco sensor de corrente

#define BAT_SENSOR_LL "xxxxxxx" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21249" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21250" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21251" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "xx:xx:xx:xx:xx:xx"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:26:34" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:26:2c"  //sucção ckt 1

#endif
```

10026

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxx" procurar

#define VOLT_CAL  244.53 //calibracao do sensor de tensao

#define VOLT_CAL_S  180 //calibracao do sensor de tensao

#define VOLT_CAL_TT  180 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 36   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "10026"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0578"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91526"  //endereco sensor de insuflamento

#define RET_SENSOR "s91527"  //endereco sensor de retorno

#define SUC_SENSOR "s91528"  //endereco sensor de sucção

#define LL_SENSOR "s91529"   //endereco sensor de linha de liquido

//#define EXT_SENSOR "sTesteExt"  //endereco sensor de externa

#define ENT_CONDES "s91539"  //endereco sensor de externa

#define SAD_CONDES "xxxxxx"  //

#define VOLT_SENSOR_R "s91530" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91531" //endereco sensor de corrente

#define CURR_SENSOR_S "xxxxx" //endereco sensor de corrente

#define CURR_SENSOR_TT "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_INS "s91535" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91536" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91537" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91538" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91532" //endereco sensor de corrente
```

```
#define VIBR_SENSOR_Y_SUC "s91533" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91534" //endereco sensor de corrente

#define SENS_TEMP_RET "bc:57:29:0e:2e:f4" //entrada de água

#define SENS_TEMP_INSU "bc:57:29:0e:2e:c0"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:a3" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:b7"  //sucção ckt 1

#define SENS_TEMP_ENT_CONDENS "bc:57:29:0e:19:81"  //externa

#define SENS_TEMP_SAD_CONDENS "bc:57:29:0e:xx:xx"  //endereço para a leitura do sensor de saida da condensação

#endif
```

10108

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  244.2 //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  17.22 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "10108"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0538"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21297"  //endereco sensor de saida

#define RET_SENSOR "s21298"  //endereco sensor de retorno

#define SUC_SENSOR "s21299"  //endereco sensor de sucção

#define LL_SENSOR "s21300"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21301" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s21302" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21306" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21307" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21308" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21309" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21303" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21304" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21305" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 2e: b1" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 26: 09"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 96" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 6e"  //sucção ckt 1

#endif
```

10254

```
#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 = "xxxxxxxxxxxxxxxxxxxxx";
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             32   // Corrente R
#define PIN_CURR_S             36  // Corrente S
#define PIN_CURR_T             39   // Corrente T

#define PIN_VOLT_R             33   // Tensão R
#define PIN_VOLT_S             34   // Tensão S
#define PIN_VOLT_T             35   // 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 191.34 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL2 179.68 //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define VOLT_CAL3 177.89  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL1 14.25   //17.7VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL2 13.04  //VALOR DE CALIBRAÇÃO (DEVE SER AJUSTADO EM PARALELO COM UM
MULTÍMETRO)
#define CURR_CAL3 16.41  //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:2e:f2", //Insuflamento
    "bc:57:29:0e:2e:c4", //Retorno
    "bc:57:29:0e:19:71", // Sucção
    "bc:57:29:0e:19:cc", // 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["s91344"] = Irms1;
doc1["s91345"] = Irms2;
doc1["s91346"] = Irms3;
doc1["s91341"] = Vrms4;
doc1["s91342"] = Vrms5;
doc1["s91343"] = Vrms6;
doc1["s91347"] = BLEX;



String STD1;

JsonDocument doc2;
doc2["t"] = timeStemp;
doc2["s91337"] = BLEIns;
doc2["s91338"] = BLERet;
doc2["s91339"] = BLESuc;
doc2["s91340"] = BLELinha;
//doc2["BLEext"] = BLEExt;
doc2["s91348"] = BLEY;
doc2["s91349"] = BLEZ;
```

```
String STD2;

JsonDocument doc3;
doc3["t"] = timeStemp;
doc3["s91350"] = Bat_ins;
doc3["s91351"] = Bat_ret;
doc3["s91352"] = Bat_suc;
doc3["s91353"] = 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/matr0540",mensa1); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0540",mensa2); // Envio de dados para determinado lugar do tópico
    delay (1000);
    MQTT.publish("v4/matr0540",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("10254", "matr4", "canudos92sc"))
    {
        Serial.println("* Tentando se conectar ao Broker MQTT: ");
        if (MQTT.connect("10254", "matr4", "canudos92sc"))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
            MQTT.subscribe("/v4/matr0540");
        }
        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;
```

```
  }
}



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

12837

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  495.52 //calibracao do sensor de tensao

#define VOLT_CAL_S  10 //calibracao do sensor de tensao

#define VOLT_CAL_TT  10 //calibracao do sensor de tensao

#define CURRENT_CAL  17.69 //calibracao do sensor de corrente

#define CURRENT_CAL_S 10  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  10 //calibracao do sensor de corrente

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 32   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  39 //pino para a leitura da tensao  PIN_VOLTAGE


#define EQUIPAMENT_TAG "12837"  //tag do equipamento, deve ser fornecido pelo spo
```

```
EQUIPAMENT_TAG

#define EQUIPAMENT_TOPIC "v4/matr0531"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21178"  //endereco sensor de saida

#define RET_SENSOR "s21179"  //endereco sensor de retorno

#define SUC_SENSOR "s21180"  //endereco sensor de sucção

#define LL_SENSOR "s21181"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21182" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s21183" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21187" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21188" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21189" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21190" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21184" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21185" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21186" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 98" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: 4b"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 95" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 3c"  //sucção ckt 1

#endif
```

12838

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  509.22  //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  17.24 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "12838"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0533"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21221"  //endereco sensor de saida

#define RET_SENSOR "s21222"  //endereco sensor de retorno

#define SUC_SENSOR "s21223"  //endereco sensor de sucção

#define LL_SENSOR "s21224"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21225" //endereco sensor de tensao
#define CURR_SENSOR_R "s21226" //endereco sensor de corrente ckt 1

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao
#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao
#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1
#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21230" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21231" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21232" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21233" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21227" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21228" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21229" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: df" //entrada de água

#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: d1"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 25: f2" //descarga ckt 1
```

```
#define SENS_TEMP_SUC "bc:57:29:0e:19:ba"   //sucção ckt 1


#endif
```

12842

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  113.42 //calibracao do sensor de tensao

#define VOLT_CAL_S  108.7 //calibracao do sensor de tensao

#define VOLT_CAL_TT  506.14 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  13.125 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39   //pino para a leitura de corrente

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "12842"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0517"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20949"  //endereco sensor de saida

#define RET_SENSOR "s20950"  //endereco sensor de retorno

#define SUC_SENSOR "s20951"  //endereco sensor de sucção

#define LL_SENSOR "s20952"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20953" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20962" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20963" //endereco sensor de tensao

#define CURR_SENSOR_R "s20954" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20964" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20965" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s20958" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20959" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20960" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20961" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20955" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20956" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20957" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 2b" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 19: 75"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 26: 00" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 88"  //sucção ckt 1

#endif
```

12845

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  228.9  //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  17.27 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "12845"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0523"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21046"  //endereco sensor de saida

#define RET_SENSOR "s21047"  //endereco sensor de retorno

#define SUC_SENSOR "s21048"  //endereco sensor de sucção

#define LL_SENSOR "s21049"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21050" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s21051" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21055" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21056" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21057" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21058" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21052" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21053" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21054" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 2e: f9" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 25: fc"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: c9" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 26: 27"  //sucção ckt 1

#endif
```

13132 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  72.59 //calibracao do sensor de tensao

#define VOLT_CAL_S  68 //calibracao do sensor de tensao

#define VOLT_CAL_TT  80 //calibracao do sensor de tensao

#define CURRENT_CAL  15.42 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.05  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  16.33 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "13132"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0536"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "xxxxxx"  //endereco sensor de saida

#define RET_SENSOR "xxxxxxx"  //endereco sensor de retorno

#define SUC_SENSOR "s21265"  //endereco sensor de sucção

#define LL_SENSOR "s21266"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s21281" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21282" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21283" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "xxxxx" //endereco sensor de corrente

#define BAT_SENSOR_RET "xxxxxx" //endereco sensor de corrente

#define BAT_SENSOR_SUC "xxxxxx" //endereco sensor de corrente

#define BAT_SENSOR_LL "xxxxxxx" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21274" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21275" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21276" //endereco sensor de corrente

#define SENS_TEMP_RET "xx: xx: xx: xx: xx: xx" //entrada de água
```

```
#define SENS_TEMP_INSU "xx:xx:xx:xx:xx:xx"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:be" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:2e:df"  //sucção ckt 1

#endif
```

13318

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx#" procurar

#define VOLT_CAL  181.62  //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  24.27 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "13318"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0520"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20991"  //endereco sensor de saida

#define RET_SENSOR "s20992"  //endereco sensor de retorno

#define SUC_SENSOR "s20993"  //endereco sensor de sucção

#define LL_SENSOR "s20994"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20995" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s20996" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21000" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21001" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21002" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21003" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20997" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20998" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20999" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 9e" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:2e:f6"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:b3" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:37"  //sucção ckt 1

#endif
```

17406

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

#define VOLT_CAL  81.8 //calibracao do sensor de tensao

#define VOLT_CAL_S  80.91 //calibracao do sensor de tensao

#define VOLT_CAL_TT  76.55 //calibracao do sensor de tensao

#define CURRENT_CAL  18.08 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.38  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.8 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "17406"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0532"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21204"  //endereco sensor de saida

#define RET_SENSOR "s21205"  //endereco sensor de retorno

#define SUC_SENSOR "s21206"  //endereco sensor de sucção

#define LL_SENSOR "s21207"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21208" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21209" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21210" //endereco sensor de tensao

#define CURR_SENSOR_R "s21211" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21212" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21213" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21217" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21218" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21219" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21220" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21214" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21215" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21216" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: 79" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 2e: d7"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 5d" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: dc"  //sucção ckt 1

#endif
```

17407

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

#define VOLT_CAL  79.21 //calibracao do sensor de tensao

#define VOLT_CAL_S  80.81 //calibracao do sensor de tensao

#define VOLT_CAL_TT  75.66 //calibracao do sensor de tensao

#define CURRENT_CAL  18.03 //calibracao do sensor de corrente

#define CURRENT_CAL_S 16.16  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14.54 //calibracao do sensor de corrente

#define PIN_CURRENT  39 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //pino para a leitura de corrente

#define PIN_CURRENT_TT 35   //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 32  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  36 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "17407"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0516"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s20890"  //endereco sensor de saida

#define RET_SENSOR "s20891"  //endereco sensor de retorno

#define SUC_SENSOR "s20892"  //endereco sensor de sucção

#define LL_SENSOR "s20893"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s20894" //endereco sensor de tensao

#define VOLT_SENSOR_S "s20895" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s20896" //endereco sensor de tensao

#define CURR_SENSOR_R "s20897" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s20898" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s20899" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s20903" //endereco sensor de corrente

#define BAT_SENSOR_RET "s20904" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s20905" //endereco sensor de corrente

#define BAT_SENSOR_LL "s20906" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s20900" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s20901" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s20902" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 2e: d2" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:2f:17"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:53" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:aa"  //sucção ckt 1

#endif
```

23916

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  354.03  //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "23916"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0544"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91433"  //endereco sensor de saida

#define RET_SENSOR "s91434"  //endereco sensor de retorno

#define SUC_SENSOR "s91435"  //endereco sensor de sucção

#define LL_SENSOR "s91436"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s91437" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91438" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s91442" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91443" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91444" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91445" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91439" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s91440" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91441" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 26: 0c" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 2e: ff"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 87" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 19: 64"  //sucção ckt 1

#endif
```

36512

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  174.38 //calibracao do sensor de tensao

#define VOLT_CAL_S  193 //calibracao do sensor de tensao

#define VOLT_CAL_TT  187.33 //calibracao do sensor de tensao

#define CURRENT_CAL  14.32 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14.21  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  15.28 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39    //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "36512"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0529"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21144"  //endereco sensor de saida

#define RET_SENSOR "s21145"  //endereco sensor de retorno

#define SUC_SENSOR "s21146"  //endereco sensor de sucção

#define LL_SENSOR "s21147"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21148" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21157" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21158" //endereco sensor de tensao

#define CURR_SENSOR_R "s21149" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21159" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21160" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21153" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21154" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21155" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21156" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21150" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21151" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21152" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 26: 29" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:26:26"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:b5" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:26:01"  //sucção ckt 1

#endif
```

36514

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  195.62 //calibracao do sensor de tensao

#define VOLT_CAL_S  176.52 //calibracao do sensor de tensao

#define VOLT_CAL_TT  178.58 //calibracao do sensor de tensao

#define CURRENT_CAL  14 //calibracao do sensor de corrente

#define CURRENT_CAL_S 14  //calibracao do sensor de corrente

#define CURRENT_CAL_TT  14 //calibracao do sensor de corrente

#define PIN_CURRENT  32 //pino para a leitura de corrente

#define PIN_CURRENT_S 36  //pino para a leitura de corrente

#define PIN_CURRENT_TT 39    //pino para a leitura de corrente

#define PIN_TEMPE 16    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 33  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  34 //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_TT  35 //pino para a leitura da tensao  PIN_VOLTAGE

#define EQUIPAMENT_TAG "36514"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0525"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s21076"  //endereco sensor de saida

#define RET_SENSOR "s21077"  //endereco sensor de retorno

#define SUC_SENSOR "s21078"  //endereco sensor de sucção

#define LL_SENSOR "s21079"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s21080" //endereco sensor de tensao

#define VOLT_SENSOR_S "s21089" //endereco sensor de tensao

#define VOLT_SENSOR_TT "s21090" //endereco sensor de tensao

#define CURR_SENSOR_R "s21081" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "s21091" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "s21092" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s21085" //endereco sensor de corrente

#define BAT_SENSOR_RET "s21086" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s21087" //endereco sensor de corrente

#define BAT_SENSOR_LL "s21088" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s21082" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s21083" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s21084" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 19: e2" //entrada de água
```

```
#define SENS_TEMP_INSU "bc:57:29:0e:19:86"  // saída de água

#define SENS_TEMP_LL "bc:57:29:0e:19:b6" //descarga ckt 1

#define SENS_TEMP_SUC "bc:57:29:0e:19:39"  //sucção ckt 1

#endif
```

36627

```
#ifndef _ENV_H
#define _ENV_H

#define NETWORK_CLIENT "SmartVac Telemetria"

#define PASSW  "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" procurar

#define VOLT_CAL  165.09 //calibracao do sensor de tensao

#define VOLT_CAL_S  0000  //ignorar

#define VOLT_CAL_TT  00000  //ignorar

#define CURRENT_CAL  16.32 //calibracao do sensor de corrente

#define CURRENT_CAL_S 0000  //ignorar

#define CURRENT_CAL_TT  0000000 //ignorar

#define PIN_CURRENT  34 //pino para a leitura de corrente

#define PIN_CURRENT_S 33  //ignorar

#define PIN_CURRENT_TT 36   //ignorar

#define PIN_TEMPE 15    //pino para a leitura temperatura  D5 no hard

#define PIN_VOLTAGE 35  //pino para a leitura da tensao  PIN_VOLTAGE

#define PIN_VOLTAGE_S  32 //ignorar

#define PIN_VOLTAGE_TT  39 //ignorar

#define EQUIPAMENT_TAG "36627"  //tag do equipamento, deve ser fornecido pelo spo
EQUIPAMENT_TAG
```

```
#define EQUIPAMENT_TOPIC "v4/matr0545"  //topico de envio, fornecido pelo spo
EQUIPAMENT_TOPIC

#define INS_SENSOR "s91446"  //endereco sensor de saida

#define RET_SENSOR "s91447"  //endereco sensor de retorno

#define SUC_SENSOR "s91448"  //endereco sensor de sucção

#define LL_SENSOR "s91449"   //endereco sensor de linha de liquido

#define VOLT_SENSOR_R "s91450" //endereco sensor de tensao

#define VOLT_SENSOR_S "xxxxxxx" //endereco sensor de tensao

#define VOLT_SENSOR_TT "xxxxxx" //endereco sensor de tensao

#define CURR_SENSOR_R "s91451" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_S "xxxxxx" //endereco sensor de corrente ckt 1

#define CURR_SENSOR_TT "xxxxxx" //endereco sensor de corrente ckt 1

#define BAT_SENSOR_INS "s91455" //endereco sensor de corrente

#define BAT_SENSOR_RET "s91456" //endereco sensor de corrente

#define BAT_SENSOR_SUC "s91457" //endereco sensor de corrente

#define BAT_SENSOR_LL "s91458" //endereco sensor de corrente

#define VIBR_SENSOR_X_SUC "s91452" //endereco sensor de corrente

#define VIBR_SENSOR_Y_SUC "s91453" //endereco sensor de corrente

#define VIBR_SENSOR_Z_SUC "s91454" //endereco sensor de corrente

#define SENS_TEMP_RET "bc: 57: 29: 0e: 25: f5" //entrada de água
```

```
#define SENS_TEMP_INSU "bc: 57: 29: 0e: 2e: b8"  // saída de água

#define SENS_TEMP_LL "bc: 57: 29: 0e: 19: 82" //descarga ckt 1

#define SENS_TEMP_SUC "bc: 57: 29: 0e: 26: 28"  //sucção ckt 1

#endif
```

9575

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9575";
    constexpr char TOPIC[] = "v4/matr0569";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  80.57;
    static constexpr float CAL_S = 81.87;
    static constexpr float CAL_TT = 79.3;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 18.83;
    static constexpr int CAL_S = 19.17;
    static constexpr int CAL_TT =  19.09;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000243";
    constexpr char RET[] = "s100000244";
    constexpr char SUC[] = "s100000245";
    constexpr char LL[] = "s100000246";
    constexpr char ENT_CONDES[] = "s100000260";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000247";
    constexpr char VOLT_S[] = "s100000256";
    constexpr char VOLT_TT[] = "s100000257";

    constexpr char CURR_R[] = "s100000248";
    constexpr char CURR_S[] = "s100000258";
    constexpr char CURR_TT[] = "s100000259";

    constexpr char BAT_INS[] = "s100000252";
    constexpr char BAT_RET[] = "s100000253";
    constexpr char BAT_SUC[] = "s100000254";
    constexpr char BAT_LL[] = "s100000255";

    constexpr char VIBR_X_SUC[] = "s100000249";
    constexpr char VIBR_Y_SUC[] = "s100000250";
    constexpr char VIBR_Z_SUC[] = "s100000251";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:c8"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:13";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:7f";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:80";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c6";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9824 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9824";
    constexpr char TOPIC[] = "v4/matr0568b";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  214.28;
    static constexpr float CAL_S = 226.59;
    static constexpr float CAL_TT = 101.09;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.39;
    static constexpr int CAL_S = 15.84;
    static constexpr int CAL_TT = 13.71;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "xxxx";
    constexpr char RET[] = "xxx";
    constexpr char SUC[] = "s100000220";
    constexpr char LL[] = "s100000221";
    constexpr char ENT_CONDES[] = "s100000239";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "xxx";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s100000236";
    constexpr char CURR_S[] = "s100000237";
    constexpr char CURR_TT[] = "s100000238";

    constexpr char BAT_INS[] = "xxx";
    constexpr char BAT_RET[] = "xxx";
    constexpr char BAT_SUC[] = "xxx";
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s100000229";
    constexpr char VIBR_Y_SUC[] = "s100000230";
    constexpr char VIBR_Z_SUC[] = "s100000231";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:xx:xx"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:xx:xx";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:e8";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:fd";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:5c";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9824 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9824";
    constexpr char TOPIC[] = "v4/matr0568";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = xxxx;
    static constexpr float CAL_S = xxx;
    static constexpr float CAL_TT = xxx;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = xxx;
    static constexpr int CAL_S = xxx;
    static constexpr int CAL_TT = xxx;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000214";
    constexpr char RET[] = "s100000215";
    constexpr char SUC[] = "s100000218";
    constexpr char LL[] = "s100000220";
    constexpr char ENT_CONDES[] = "s100000239";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000240";
    constexpr char VOLT_S[] = "s100000241";
    constexpr char VOLT_TT[] = "s100000242";

    constexpr char CURR_R[] = "s100000217";
    constexpr char CURR_S[] = "s100000224";
    constexpr char CURR_TT[] = "s100000225";

    constexpr char BAT_INS[] = "s100000232";
    constexpr char BAT_RET[] = "s100000233";
    constexpr char BAT_SUC[] = "s100000234";
    constexpr char BAT_LL[] = "s100000235";

    constexpr char VIBR_X_SUC[] = "s100000226";
    constexpr char VIBR_Y_SUC[] = "s100000227";
    constexpr char VIBR_Z_SUC[] = "s100000228";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:xx:xx"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:xx:xx";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:xx:xx";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:xx:xx";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:xx:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9571

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9571";
    constexpr char TOPIC[] = "v4/matr0567";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  424.03;
    static constexpr float CAL_S = 125.77;
    static constexpr float CAL_TT = 116;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.53;
    static constexpr int CAL_S = 14.75;
    static constexpr int CAL_TT = 14.37;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000196";
    constexpr char RET[] = "s100000197";
    constexpr char SUC[] = "s100000198";
    constexpr char LL[] = "s100000199";
    constexpr char ENT_CONDES[] = "s100000213";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000200";
    constexpr char VOLT_S[] = "s100000209";
    constexpr char VOLT_TT[] = "s100000210";

    constexpr char CURR_R[] = "s100000201";
    constexpr char CURR_S[] = "s100000211";
    constexpr char CURR_TT[] = "s100000212";

    constexpr char BAT_INS[] = "s100000205";
    constexpr char BAT_RET[] = "s100000206";
    constexpr char BAT_SUC[] = "s100000207";
    constexpr char BAT_LL[] = "s100000208";

    constexpr char VIBR_X_SUC[] = "s100000202";
    constexpr char VIBR_Y_SUC[] = "s100000203";
    constexpr char VIBR_Z_SUC[] = "s100000204";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:87"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:21";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:f8";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:7d";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c6";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10246

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10246";
    constexpr char TOPIC[] = "v4/matr0565";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  162.62;
    static constexpr float CAL_S = 313.56;
    static constexpr float CAL_TT = 103.93;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S =  14;
    static constexpr int CAL_TT = 14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000178";
    constexpr char RET[] = "s100000179";
    constexpr char SUC[] = "s100000180";
    constexpr char LL[] = "s100000181";
    constexpr char ENT_CONDES[] = "s100000195";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000182";
    constexpr char VOLT_S[] = "s100000191";
    constexpr char VOLT_TT[] = "s100000192";

    constexpr char CURR_R[] = "s100000183";
    constexpr char CURR_S[] = "s100000193";
    constexpr char CURR_TT[] = "s100000194";

    constexpr char BAT_INS[] = "s100000187";
    constexpr char BAT_RET[] = "s100000188";
    constexpr char BAT_SUC[] = "s100000189";
    constexpr char BAT_LL[] = "s100000190";

    constexpr char VIBR_X_SUC[] = "s100000184";
    constexpr char VIBR_Y_SUC[] = "s100000185";
    constexpr char VIBR_Z_SUC[] = "s100000186";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:51"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:59";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:13";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:e6";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:54";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10243

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10243";
    constexpr char TOPIC[] = "v4/matr0564";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  491.29;
    static constexpr float CAL_S = 103.06;
    static constexpr float CAL_TT = 104.82;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13;
    static constexpr int CAL_S =  13.17;
    static constexpr int CAL_TT = 13.5;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000160";
    constexpr char RET[] = "s100000161";
    constexpr char SUC[] = "s100000162";
    constexpr char LL[] = "s100000163";
    constexpr char ENT_CONDES[] = "s100000177";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000164";
    constexpr char VOLT_S[] = "s100000173";
    constexpr char VOLT_TT[] = "s100000174";

    constexpr char CURR_R[] = "s100000165";
    constexpr char CURR_S[] = "s100000175";
    constexpr char CURR_TT[] = "s100000176";

    constexpr char BAT_INS[] = "s100000169";
    constexpr char BAT_RET[] = "s100000170";
    constexpr char BAT_SUC[] = "s100000171";
    constexpr char BAT_LL[] = "s100000172";

    constexpr char VIBR_X_SUC[] = "s100000166";
    constexpr char VIBR_Y_SUC[] = "s100000167";
    constexpr char VIBR_Z_SUC[] = "s100000168";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fc:f4"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fc:f7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:6a";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:85";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:d7";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

12388

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "12388";
    constexpr char TOPIC[] = "v4/matr0563";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  82.29;
    static constexpr float CAL_S = 83.92;
    static constexpr float CAL_TT = 76.44;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 18.92;
    static constexpr int CAL_S =  18.30;
    static constexpr int CAL_TT = 18.74;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000142";
    constexpr char RET[] = "s100000143";
    constexpr char SUC[] = "s100000144";
    constexpr char LL[] = "s100000145";
    constexpr char ENT_CONDES[] = "s100000159";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000146";
    constexpr char VOLT_S[] = "s100000155";
    constexpr char VOLT_TT[] = "s100000156";

    constexpr char CURR_R[] = "s100000147";
    constexpr char CURR_S[] = "s100000157";
    constexpr char CURR_TT[] = "s100000158";

    constexpr char BAT_INS[] = "s100000151";
    constexpr char BAT_RET[] = "s100000152";
    constexpr char BAT_SUC[] = "s100000153";
    constexpr char BAT_LL[] = "s100000154";

    constexpr char VIBR_X_SUC[] = "s100000148";
    constexpr char VIBR_Y_SUC[] = "s100000149";
    constexpr char VIBR_Z_SUC[] = "s100000150";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:b2"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:b1";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:8a";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:8c";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:ad";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9314

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9314";
    constexpr char TOPIC[] = "v4/matr0562";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  77.66;
    static constexpr float CAL_S = 77.55;
    static constexpr float CAL_TT = 68.82;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.28;
    static constexpr int CAL_S = 16.99;
    static constexpr int CAL_TT = 16.49;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s92068";
    constexpr char RET[] = "s92069";
    constexpr char SUC[] = "s92070";
    constexpr char LL[] = "s92071";
    constexpr char ENT_CONDES[] = "s92085";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s92072";
    constexpr char VOLT_S[] = "s92073";
    constexpr char VOLT_TT[] = "s92074";

    constexpr char CURR_R[] = "s92075";
    constexpr char CURR_S[] = "s92076";
    constexpr char CURR_TT[] = "s92077";

    constexpr char BAT_INS[] = "s92081";
    constexpr char BAT_RET[] = "s92082";
    constexpr char BAT_SUC[] = "s92083";
    constexpr char BAT_LL[] = "s92084";

    constexpr char VIBR_X_SUC[] = "s92078";
    constexpr char VIBR_Y_SUC[] = "s92079";
    constexpr char VIBR_Z_SUC[] = "s92080";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:94"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:86";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:ff:ff:95";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:ff:ff:93";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:88";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10245

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10245";
    constexpr char TOPIC[] = "v4/matr0561";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  79.03;
    static constexpr float CAL_S = 929.02;
    static constexpr float CAL_TT = 71.21;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 16.49;
    static constexpr int CAL_S = 16.7;
    static constexpr int CAL_TT = 16.86;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000124";
    constexpr char RET[] = "s100000125";
    constexpr char SUC[] = "s100000126";
    constexpr char LL[] = "s100000127";
    constexpr char ENT_CONDES[] = "s100000141";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000128";
    constexpr char VOLT_S[] = "s100000137";
    constexpr char VOLT_TT[] = "s100000138";

    constexpr char CURR_R[] = "s100000129";
    constexpr char CURR_S[] = "s100000139";
    constexpr char CURR_TT[] = "s100000140";

    constexpr char BAT_INS[] = "s100000133";
    constexpr char BAT_RET[] = "s100000134";
    constexpr char BAT_SUC[] = "s100000135";
    constexpr char BAT_LL[] = "s100000136";

    constexpr char VIBR_X_SUC[] = "s100000130";
    constexpr char VIBR_Y_SUC[] = "s100000131";
    constexpr char VIBR_Z_SUC[] = "s100000132";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fc:f4"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fc:f7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:6a";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:85";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:d7";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

36516

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "36516";
    constexpr char TOPIC[] = "v4/matr0560";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  174.54;
    static constexpr float CAL_S = 176.09;
    static constexpr float CAL_TT = 172.11;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.46;
    static constexpr int CAL_S = 13.75;
    static constexpr int CAL_TT = 15.9;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000106";
    constexpr char RET[] = "s100000107";
    constexpr char SUC[] = "s100000108";
    constexpr char LL[] = "s100000109";
    constexpr char ENT_CONDES[] = "s100000123";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000110";
    constexpr char VOLT_S[] = "s100000119";
    constexpr char VOLT_TT[] = "s100000120";

    constexpr char CURR_R[] = "s100000111";
    constexpr char CURR_S[] = "s100000121";
    constexpr char CURR_TT[] = "s100000122";

    constexpr char BAT_INS[] = "s100000115";
    constexpr char BAT_RET[] = "s100000116";
    constexpr char BAT_SUC[] = "s100000117";
    constexpr char BAT_LL[] = "s100000118";

    constexpr char VIBR_X_SUC[] = "s100000112";
    constexpr char VIBR_Y_SUC[] = "s100000113";
    constexpr char VIBR_Z_SUC[] = "s100000114";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:c5"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:ab";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:78";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:89";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:aa";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10244

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10244";
    constexpr char TOPIC[] = "v4/matr0559";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  206.8;
    static constexpr float CAL_S = 220.83;
    static constexpr float CAL_TT = 1051;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT = 14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000088";
    constexpr char RET[] = "s100000089";
    constexpr char SUC[] = "s100000090";
    constexpr char LL[] = "s100000091";
    constexpr char ENT_CONDES[] = "s100000105";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000092";
    constexpr char VOLT_S[] = "s100000101";
    constexpr char VOLT_TT[] = "s100000102";

    constexpr char CURR_R[] = "s100000093";
    constexpr char CURR_S[] = "s100000103";
    constexpr char CURR_TT[] = "s100000104";

    constexpr char BAT_INS[] = "s100000097";
    constexpr char BAT_RET[] = "s100000098";
    constexpr char BAT_SUC[] = "s100000099";
    constexpr char BAT_LL[] = "s100000100";

    constexpr char VIBR_X_SUC[] = "s100000094";
    constexpr char VIBR_Y_SUC[] = "s100000095";
    constexpr char VIBR_Z_SUC[] = "s100000096";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:c4"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:b3";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:72";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:6b";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:d7";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9825 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9825";
    constexpr char TOPIC[] = "v4/matr0558";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 79.11;
    static constexpr float CAL_S =  79.88;
    static constexpr float CAL_TT = 939.88;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.24;
    static constexpr int CAL_S = 14.35;
    static constexpr int CAL_TT = 14.55;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000062";
    constexpr char RET[] = "s100000063";
    constexpr char SUC[] = "s100000068";
    constexpr char LL[] = "s100000069";
    constexpr char ENT_CONDES[] = "s100000087";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "xxxxxxxxxxx";
    constexpr char VOLT_S[] = "xxxxxxxxxxx";
    constexpr char VOLT_TT[] = "xxxxxxxxxxxxxxx";

    constexpr char CURR_R[] = "s100000084";
    constexpr char CURR_S[] = "s100000085";
    constexpr char CURR_TT[] = "s100000086";

    constexpr char BAT_INS[] = "xxxxxxxxxxxxxxxxxx";
    constexpr char BAT_RET[] = "xxxxxxxxxxxxxxxx";
    constexpr char BAT_SUC[] = "xxxxxxxxxxxxxxx";
    constexpr char BAT_LL[] = "xxxxxxxxxxxxxx";

    constexpr char VIBR_X_SUC[] = "s100000078";
    constexpr char VIBR_Y_SUC[] = "s100000079";
    constexpr char VIBR_Z_SUC[] = "s100000080";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:77"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:4f";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:2e";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fd:9c";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:5c";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9428

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9428";
    constexpr char TOPIC[] = "v4/matr0557";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  177.45;
    static constexpr float CAL_S =  176.21;
    static constexpr float CAL_TT =  177.27;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R =  13.41;
    static constexpr int CAL_S = 13.83;
    static constexpr int CAL_TT = 13.32;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000036";
    constexpr char RET[] = "s100000037";
    constexpr char SUC[] = "S100000042";
    constexpr char LL[] = "S100000043";
    constexpr char ENT_CONDES[] = "s100000061";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "XXXXX";
    constexpr char VOLT_S[] = "XXXXX";
    constexpr char VOLT_TT[] = "XXXXXX";

    constexpr char CURR_R[] = "S100000058";
    constexpr char CURR_S[] = "S100000059";
    constexpr char CURR_TT[] = "S100000060";

    constexpr char BAT_INS[] = "XXXXX";
    constexpr char BAT_RET[] = "XXXXXX";
    constexpr char BAT_SUC[] = "XXXXXX";
    constexpr char BAT_LL[] = "XXXXX";

    constexpr char VIBR_X_SUC[] = "S100000051";
    constexpr char VIBR_Y_SUC[] = "S100000052";
    constexpr char VIBR_Z_SUC[] = "S100000053";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:58"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:57";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:ef";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:eb";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:56";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9825 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9825";
    constexpr char TOPIC[] = "v4/matr0558";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 236.78;
    static constexpr float CAL_S =  101;
    static constexpr float CAL_TT = 212.77;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.8;
    static constexpr int CAL_S = 13.5;
    static constexpr int CAL_TT = 14.13;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000062";
    constexpr char RET[] = "s100000063";
    constexpr char SUC[] = "s100000066";
    constexpr char LL[] = "s100000067";
    constexpr char ENT_CONDES[] = "s100000087";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000064";
    constexpr char VOLT_S[] = "s100000070";
    constexpr char VOLT_TT[] = "s100000071";

    constexpr char CURR_R[] = "s100000065";
    constexpr char CURR_S[] = "s100000072";
    constexpr char CURR_TT[] = "s100000073";

    constexpr char BAT_INS[] = "s100000080";
    constexpr char BAT_RET[] = "s100000081";
    constexpr char BAT_SUC[] = "s100000082";
    constexpr char BAT_LL[] = "s100000083";

    constexpr char VIBR_X_SUC[] = "s100000074";
    constexpr char VIBR_Y_SUC[] = "s100000075";
    constexpr char VIBR_Z_SUC[] = "s100000076";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:77"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:4f";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:e2";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:09";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:5c";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9428 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9428";
    constexpr char TOPIC[] = "v4/matr0557";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  177.45;
    static constexpr float CAL_S =  176.21;
    static constexpr float CAL_TT =  177.27;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R =  13.14;
    static constexpr int CAL_S = 13.06;
    static constexpr int CAL_TT = 13.46;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000036";
    constexpr char RET[] = "s100000037";
    constexpr char SUC[] = "s100000040";
    constexpr char LL[] = "s100000041";
    constexpr char ENT_CONDES[] = "s100000061";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000038";
    constexpr char VOLT_S[] = "s100000044";
    constexpr char VOLT_TT[] = "s100000045";

    constexpr char CURR_R[] = "s100000039";
    constexpr char CURR_S[] = "s100000046";
    constexpr char CURR_TT[] = "s100000047";

    constexpr char BAT_INS[] = "s100000054";
    constexpr char BAT_RET[] = "s100000055";
    constexpr char BAT_SUC[] = "s100000056";
    constexpr char BAT_LL[] = "s100000057";

    constexpr char VIBR_X_SUC[] = "s100000048";
    constexpr char VIBR_Y_SUC[] = "s100000049";
    constexpr char VIBR_Z_SUC[] = "s100000050";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:58"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:57";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:e5";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:ec";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:56";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

36513

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "36513";
    constexpr char TOPIC[] = "v4/matr0553";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  169.46;
    static constexpr float CAL_S =  194.06;
    static constexpr float CAL_TT = 187.86;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.36;
    static constexpr int CAL_S = 13.73;
    static constexpr int CAL_TT = 14.54;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91979";
    constexpr char RET[] = "s91980";
    constexpr char SUC[] = "s91981";
    constexpr char LL[] = "s91982";
    constexpr char ENT_CONDES[] = "s91996";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91983";
    constexpr char VOLT_S[] = "s91992";
    constexpr char VOLT_TT[] = "s91993";

    constexpr char CURR_R[] = "s91984";
    constexpr char CURR_S[] = "s91994";
    constexpr char CURR_TT[] = "s91995";

    constexpr char BAT_INS[] = "s91988";
    constexpr char BAT_RET[] = "s91989";
    constexpr char BAT_SUC[] = "s91990";
    constexpr char BAT_LL[] = "s91991";

    constexpr char VIBR_X_SUC[] = "s91985";
    constexpr char VIBR_Y_SUC[] = "s91986";
    constexpr char VIBR_Z_SUC[] = "s91987";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fe:b8"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:87";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:2f";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:31";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:ff:3b";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

17336

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "17336";
    constexpr char TOPIC[] = "v4/matr0556";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  424.03;
    static constexpr float CAL_S = 125.77;
    static constexpr float CAL_TT = 116;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.53;
    static constexpr int CAL_S = 14.75;
    static constexpr int CAL_TT = 14.37;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000018";
    constexpr char RET[] = "s100000019";
    constexpr char SUC[] = "s100000020";
    constexpr char LL[] = "s100000021";
    constexpr char ENT_CONDES[] = "s100000035";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000022";
    constexpr char VOLT_S[] = "s100000031";
    constexpr char VOLT_TT[] = "s100000032";

    constexpr char CURR_R[] = "s100000023";
    constexpr char CURR_S[] = "s100000033";
    constexpr char CURR_TT[] = "s100000034";

    constexpr char BAT_INS[] = "s100000027";
    constexpr char BAT_RET[] = "s100000028";
    constexpr char BAT_SUC[] = "s100000029";
    constexpr char BAT_LL[] = "s100000030";

    constexpr char VIBR_X_SUC[] = "s100000024";
    constexpr char VIBR_Y_SUC[] = "s100000025";
    constexpr char VIBR_Z_SUC[] = "s100000026";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:5a"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:5d";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:18";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:2a";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:55";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

36515

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "36515";
    constexpr char TOPIC[] = "v4/matr0555";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  174.54;
    static constexpr float CAL_S = 176.09;
    static constexpr float CAL_TT = 172.11;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.46;
    static constexpr int CAL_S = 13.75;
    static constexpr int CAL_TT = 15.9;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000000";
    constexpr char RET[] = "s100000001";
    constexpr char SUC[] = "s100000002";
    constexpr char LL[] = "s100000003";
    constexpr char ENT_CONDES[] = "s100000017";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000004";
    constexpr char VOLT_S[] = "s100000013";
    constexpr char VOLT_TT[] = "s100000014";

    constexpr char CURR_R[] = "s100000005";
    constexpr char CURR_S[] = "s100000015";
    constexpr char CURR_TT[] = "s100000016";

    constexpr char BAT_INS[] = "s100000009";
    constexpr char BAT_RET[] = "s100000010";
    constexpr char BAT_SUC[] = "s100000011";
    constexpr char BAT_LL[] = "s100000012";

    constexpr char VIBR_X_SUC[] = "s100000006";
    constexpr char VIBR_Y_SUC[] = "s100000007";
    constexpr char VIBR_Z_SUC[] = "s100000008";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:78"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:84";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:fb";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:01";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:50";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9736 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9736";
    constexpr char TOPIC[] = "v4/matr0561";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  175.18;
    static constexpr float CAL_S = 174.09;
    static constexpr float CAL_TT = 171.36;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.8;
    static constexpr int CAL_S = 13.8;
    static constexpr int CAL_TT = 13.8;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s92065";
    constexpr char RET[] = "s92045";
    constexpr char SUC[] = "s92053";
    constexpr char LL[] = "s92054";
    constexpr char ENT_CONDES[] = "s92046";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "XXX";
    constexpr char VOLT_S[] = "XXX";
    constexpr char VOLT_TT[] = "XXX";

    constexpr char CURR_R[] = "s92047";
    constexpr char CURR_S[] = "s92048";
    constexpr char CURR_TT[] = "s92049";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxxx";
    constexpr char BAT_LL[] = "xxxxx";

    constexpr char VIBR_X_SUC[] = "s92058";
    constexpr char VIBR_Y_SUC[] = "s92059";
    constexpr char VIBR_Z_SUC[] = "s92060";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:b6"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:af";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:70";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:88";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:2e:b4";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9441 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9441";
    constexpr char TOPIC[] = "v4/matr0550";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  199.23;
    static constexpr float CAL_S = 199.96;
    static constexpr float CAL_TT = 1047.7;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 12.57;
    static constexpr int CAL_S = 13.87;
    static constexpr int CAL_TT = 12.43;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91953";
    constexpr char RET[] = "s91933";
    constexpr char SUC[] = "s91954";
    constexpr char LL[] = "s91955";
    constexpr char ENT_CONDES[] = "s91934";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91943";
    constexpr char VOLT_S[] = "s91944";
    constexpr char VOLT_TT[] = "s91945";

    constexpr char CURR_R[] = "s91938";
    constexpr char CURR_S[] = "s91939";
    constexpr char CURR_TT[] = "s91940";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxx";
    constexpr char BAT_LL[] = "xxxx";

    constexpr char VIBR_X_SUC[] = "s91949";
    constexpr char VIBR_Y_SUC[] = "s91950";
    constexpr char VIBR_Z_SUC[] = "s91951";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:c2"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:a4";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:00";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:0f";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c7";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9441 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9441";
    constexpr char TOPIC[] = "v4/matr0550";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  199.23;
    static constexpr float CAL_S = 199.96;
    static constexpr float CAL_TT = 1047.7;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 12.38;
    static constexpr int CAL_S = 14.18;
    static constexpr int CAL_TT = 13.21;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91953";
    constexpr char RET[] = "s91933";
    constexpr char SUC[] = "s91941";
    constexpr char LL[] = "s91942";
    constexpr char ENT_CONDES[] = "s91934";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91943";
    constexpr char VOLT_S[] = "s91944";
    constexpr char VOLT_TT[] = "s91945";

    constexpr char CURR_R[] = "s91935";
    constexpr char CURR_S[] = "s91936";
    constexpr char CURR_TT[] = "s91937";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxx";
    constexpr char BAT_LL[] = "xxxx";

    constexpr char VIBR_X_SUC[] = "s91946";
    constexpr char VIBR_Y_SUC[] = "s91947";
    constexpr char VIBR_Z_SUC[] = "s91948";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:c2"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:a4";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:32";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:1a";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c7";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9443 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9443";
    constexpr char TOPIC[] = "v4/matr0551";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  200.07;
    static constexpr float CAL_S = 147.2;
    static constexpr float CAL_TT = 169.78;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.48;
    static constexpr int CAL_S = 12.30;
    static constexpr int CAL_TT = 13.61;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s92017";
    constexpr char RET[] = "s91997";
    constexpr char SUC[] = "s92005";
    constexpr char LL[] = "s92006";
    constexpr char ENT_CONDES[] = "s91998";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s92007";
    constexpr char VOLT_S[] = "s92008";
    constexpr char VOLT_TT[] = "s92009";

    constexpr char CURR_R[] = "s91999";
    constexpr char CURR_S[] = "s92000";
    constexpr char CURR_TT[] = "s92001";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxx";
    constexpr char BAT_LL[] = "xxxx";

    constexpr char VIBR_X_SUC[] = "s92010";
    constexpr char VIBR_Y_SUC[] = "s92011";
    constexpr char VIBR_Z_SUC[] = "s92012";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:bf"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:dc";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:6d";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:8b";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c7";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9736 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9736";
    constexpr char TOPIC[] = "v4/matr0561";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  175.18;
    static constexpr float CAL_S = 174.09;
    static constexpr float CAL_TT = 171.36;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.45;
    static constexpr int CAL_S = 14.05;
    static constexpr int CAL_TT = 13.43;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s92065";
    constexpr char RET[] = "s92045";
    constexpr char SUC[] = "s92066";
    constexpr char LL[] = "s92067";
    constexpr char ENT_CONDES[] = "s92046";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s92055";
    constexpr char VOLT_S[] = "s92056";
    constexpr char VOLT_TT[] = "s92057";

    constexpr char CURR_R[] = "s92050";
    constexpr char CURR_S[] = "s92051";
    constexpr char CURR_TT[] = "92052";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxxx";
    constexpr char BAT_LL[] = "xxxxx";

    constexpr char VIBR_X_SUC[] = "s92061";
    constexpr char VIBR_Y_SUC[] = "s92062";
    constexpr char VIBR_Z_SUC[] = "s92063";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:b6"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:af";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:86";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:6f";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:2e:b4";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9439 ckt 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9439";
    constexpr char TOPIC[] = "v4/sc0005";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  174.43;
    static constexpr float CAL_S = 155.87;
    static constexpr float CAL_TT = 155.87;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 12.64;
    static constexpr int CAL_S = 13.54;
    static constexpr int CAL_TT = 13.93;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91930";
    constexpr char RET[] = "s451";
    constexpr char SUC[] = "s462";
    constexpr char LL[] = "s463";
    constexpr char ENT_CONDES[] = "s452";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s464";
    constexpr char VOLT_S[] = "s465";
    constexpr char VOLT_TT[] = "s466";

    constexpr char CURR_R[] = "s456";
    constexpr char CURR_S[] = "s457";
    constexpr char CURR_TT[] = "s458";

    constexpr char BAT_INS[] = "xxxx";
    constexpr char BAT_RET[] = "xxxxx";
    constexpr char BAT_SUC[] = "xxxx";
    constexpr char BAT_LL[] = "xxxx";

    constexpr char VIBR_X_SUC[] = "s467";
    constexpr char VIBR_Y_SUC[] = "s468";
    constexpr char VIBR_Z_SUC[] = "s469";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fe:84"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:a4";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:fe";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:ff";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c7";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9999

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9999";
    constexpr char TOPIC[] = "v4/matr0596";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 271.58;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 16;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 32;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 36;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91832";
    constexpr char RET[] = "s91833";
    constexpr char SUC[] = "s91834";
    constexpr char LL[] = "s91835";
    constexpr char ENT_CONDES[] = "s91845";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91836";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91837";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91841";
    constexpr char BAT_RET[] = "s91842";
    constexpr char BAT_SUC[] = "s91843";
    constexpr char BAT_LL[] = "s91844";

    constexpr char VIBR_X_SUC[] = "s91838";
    constexpr char VIBR_Y_SUC[] = "s91839";
    constexpr char VIBR_Z_SUC[] = "s91840";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:26:2a"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:e7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:93";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:62";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:41";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9998

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9998";
    constexpr char TOPIC[] = "v4/matr0593";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 236.53;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 16.2;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 32;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 36;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91790";
    constexpr char RET[] = "s91791";
    constexpr char SUC[] = "s91792";
    constexpr char LL[] = "s91793";
    constexpr char ENT_CONDES[] = "s91803";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91794";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91795";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91799";
    constexpr char BAT_RET[] = "s91800";
    constexpr char BAT_SUC[] = "s91801";
    constexpr char BAT_LL[] = "s91802";

    constexpr char VIBR_X_SUC[] = "s91796";
    constexpr char VIBR_Y_SUC[] = "s91797";
    constexpr char VIBR_Z_SUC[] = "s91798";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:e5"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:be";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:5c";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:25:ed";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:41";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9994

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9994";
    constexpr char TOPIC[] = "v4/matr0597";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 173.05;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.89;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 32;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 36;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91846";
    constexpr char RET[] = "s91847";
    constexpr char SUC[] = "s91848";
    constexpr char LL[] = "s91849";
    constexpr char ENT_CONDES[] = "s91859";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91850";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91851";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91855";
    constexpr char BAT_RET[] = "s91856";
    constexpr char BAT_SUC[] = "s91857";
    constexpr char BAT_LL[] = "s91858";

    constexpr char VIBR_X_SUC[] = "s91852";
    constexpr char VIBR_Y_SUC[] = "s91853";
    constexpr char VIBR_Z_SUC[] = "s91854";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:7b"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:c9";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:03";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:7e";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:41";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9955

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9955b";
    constexpr char TOPIC[] = "v4/matr0535b";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 71.90;
    static constexpr float CAL_S = 72.69;
    static constexpr float CAL_TT = 79.15;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.50;
    static constexpr int CAL_S = 17.08;
    static constexpr int CAL_TT =  16.48;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "xxxx";
    constexpr char RET[] = "xxxx";
    constexpr char SUC[] = "s21240";
    constexpr char LL[] = "s21241";
    constexpr char ENT_CONDES[] = "xxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s21236";
    constexpr char VOLT_S[] = "s21242";
    constexpr char VOLT_TT[] = "s21243";

    constexpr char CURR_R[] = "s21256";
    constexpr char CURR_S[] = "s21257";
    constexpr char CURR_TT[] = "s21258";

    constexpr char BAT_INS[] = "xxx";
    constexpr char BAT_RET[] = "xxx";
    constexpr char BAT_SUC[] = "xxx";
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s21249";
    constexpr char VIBR_Y_SUC[] = "s21250";
    constexpr char VIBR_Z_SUC[] = "s21251";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "xx:xx:29:0e:xx:xx"; //Ble2
    constexpr char TEMP_RET[] = "xx:xx:29:0e:xx:xx";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:2c";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:34";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:xx:xx:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9955 ckt 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9955";
    constexpr char TOPIC[] = "v4/matr0535a";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 182.78;
    static constexpr float CAL_S = 184.76;
    static constexpr float CAL_TT = 183.7;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT = 14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s21234";
    constexpr char RET[] = "s21235";
    constexpr char SUC[] = "s21238";
    constexpr char LL[] = "s21239";
    constexpr char ENT_CONDES[] = "xxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s21236";
    constexpr char VOLT_S[] = "s21242";
    constexpr char VOLT_TT[] = "s21243";

    constexpr char CURR_R[] = "s21237";
    constexpr char CURR_S[] = "s21244";
    constexpr char CURR_TT[] = "s21245";

    constexpr char BAT_INS[] = "s21252";
    constexpr char BAT_RET[] = "s21253";
    constexpr char BAT_SUC[] = "s21254";
    constexpr char BAT_LL[] = "s21255";

    constexpr char VIBR_X_SUC[] = "s21246";
    constexpr char VIBR_Y_SUC[] = "s21247";
    constexpr char VIBR_Z_SUC[] = "s21248";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:7c"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:56";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:9a";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:55";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9945

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9945";
    constexpr char TOPIC[] = "v4/matr0522";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 171.72;
    static constexpr float CAL_S = 183.22;
    static constexpr float CAL_TT = 185.47;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT = 14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s21029";
    constexpr char RET[] = "s21030";
    constexpr char SUC[] = "s21031";
    constexpr char LL[] = "s21032";
    constexpr char ENT_CONDES[] = "xxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s21033";
    constexpr char VOLT_S[] = "s21042";
    constexpr char VOLT_TT[] = "s21043";

    constexpr char CURR_R[] = "s21034";
    constexpr char CURR_S[] = "s21044";
    constexpr char CURR_TT[] = "s21045";

    constexpr char BAT_INS[] = "s21038";
    constexpr char BAT_RET[] = "s21039";
    constexpr char BAT_SUC[] = "s21040";
    constexpr char BAT_LL[] = "s21041";

    constexpr char VIBR_X_SUC[] = "s21035";
    constexpr char VIBR_Y_SUC[] = "s21036";
    constexpr char VIBR_Z_SUC[] = "s21037";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:fe"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:25:f9";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:4c";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:ac";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9906

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9906";
    constexpr char TOPIC[] = "v4/matr0537";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 227.87;
    static constexpr float CAL_S = 115.4;
    static constexpr float CAL_TT = 115.5;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.94;
    static constexpr int CAL_S = 10;
    static constexpr int CAL_TT = 10.6;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s21284";
    constexpr char RET[] = "s21285";
    constexpr char SUC[] = "s21286";
    constexpr char LL[] = "s21287";
    constexpr char ENT_CONDES[] = "xxxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s21289";
    constexpr char VOLT_S[] = "XXX";
    constexpr char VOLT_TT[] = "XXX";

    constexpr char CURR_R[] = "s21288";
    constexpr char CURR_S[] = "XXX";
    constexpr char CURR_TT[] = "XXX";

    constexpr char BAT_INS[] = "s21293";
    constexpr char BAT_RET[] = "s21294";
    constexpr char BAT_SUC[] = "s21295";
    constexpr char BAT_LL[] = "s21296";

    constexpr char VIBR_X_SUC[] = "s21290";
    constexpr char VIBR_Y_SUC[] = "s21291";
    constexpr char VIBR_Z_SUC[] = "s21292";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:74"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:26:25";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:38";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:8b";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9905

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9905";
    constexpr char TOPIC[] = "v4/matr0583";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 253;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 16.7;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91597";
    constexpr char RET[] = "s91598";
    constexpr char SUC[] = "s91599";
    constexpr char LL[] = "s91600";
    constexpr char ENT_CONDES[] = "s91610";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91601";
    constexpr char VOLT_S[] = "XXX";
    constexpr char VOLT_TT[] = "XXX";

    constexpr char CURR_R[] = "s91602";
    constexpr char CURR_S[] = "XXX";
    constexpr char CURR_TT[] = "XXX";

    constexpr char BAT_INS[] = "s91606";
    constexpr char BAT_RET[] = "s91607";
    constexpr char BAT_SUC[] = "s91608";
    constexpr char BAT_LL[] = "s91609";

    constexpr char VIBR_X_SUC[] = "s91603";
    constexpr char VIBR_Y_SUC[] = "s91604";
    constexpr char VIBR_Z_SUC[] = "s91605";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:de"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:d6";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:3b";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:43";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:8a";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9758

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9758";
    constexpr char TOPIC[] = "v4/matr0541";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 205.55;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.05;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91367";
    constexpr char RET[] = "s91368";
    constexpr char SUC[] = "s91369";
    constexpr char LL[] = "s91370";
    constexpr char ENT_CONDES[] = "s91380";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91371";
    constexpr char VOLT_S[] = "XXX";
    constexpr char VOLT_TT[] = "XXX";

    constexpr char CURR_R[] = "s91372";
    constexpr char CURR_S[] = "XXX";
    constexpr char CURR_TT[] = "XXX";

    constexpr char BAT_INS[] = "s91376";
    constexpr char BAT_RET[] = "s91377";
    constexpr char BAT_SUC[] = "s91378";
    constexpr char BAT_LL[] = "s91379";

    constexpr char VIBR_X_SUC[] = "s91373";
    constexpr char VIBR_Y_SUC[] = "s91374";
    constexpr char VIBR_Z_SUC[] = "s91375";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:c1"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:9d";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:25:e9";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:10";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:26:2b";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9330

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9330";
    constexpr char TOPIC[] = "v4/matr0601";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  173.80;
    static constexpr float CAL_S = 176.84;
    static constexpr float CAL_TT = 175.87;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R =  29.13;
    static constexpr int CAL_S = 30.58;
    static constexpr int CAL_TT = 29.26;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000422";
    constexpr char RET[] = "s100000423";
    constexpr char SUC[] = "s100000437"; // externa
    constexpr char LL[] = "xxx";
    constexpr char ENT_CONDES[] = "";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000427";
    constexpr char VOLT_S[] = "s100000428";
    constexpr char VOLT_TT[] = "s100000429";

    constexpr char CURR_R[] = "s100000424";
    constexpr char CURR_S[] = "s100000425";
    constexpr char CURR_TT[] = "s100000426";

    constexpr char BAT_INS[] = "s100000433";
    constexpr char BAT_RET[] = "s100000435";
    constexpr char BAT_SUC[] = "s100000436";//externa
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s100000430";
    constexpr char VIBR_Y_SUC[] = "s100000431";
    constexpr char VIBR_Z_SUC[] = "s100000432";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:22"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:ff:45";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fc:fd";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:xx:xx";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:xx:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10139 CKT 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10139";
    constexpr char TOPIC[] = "v4/matr0577";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  164.54;
    static constexpr float CAL_S = 172.91;
    static constexpr float CAL_TT = 205.07;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.4;
    static constexpr int CAL_S = 12.53;
    static constexpr int CAL_TT = 13.27;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000393";
    constexpr char RET[] = "s100000394";
    constexpr char SUC[] = "s100000399";
    constexpr char LL[] = "s100000400";
    constexpr char ENT_CONDES[] = "s100000418";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000395";
    constexpr char VOLT_S[] = "s100000401";
    constexpr char VOLT_TT[] = "s100000402";

    constexpr char CURR_R[] = "s100000415";
    constexpr char CURR_S[] = "s100000416";
    constexpr char CURR_TT[] = "s100000417";

    constexpr char BAT_INS[] = "s100000411";
    constexpr char BAT_RET[] = "s100000412";
    constexpr char BAT_SUC[] = "s100000413";
    constexpr char BAT_LL[] = "s100000414";

    constexpr char VIBR_X_SUC[] = "s100000408";
    constexpr char VIBR_Y_SUC[] = "s100000409";
    constexpr char VIBR_Z_SUC[] = "s100000410";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fe:e3"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fe:e7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:de";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:db";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fc:fb";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10139 CKT 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10139";
    constexpr char TOPIC[] = "v4/matr0577b";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  150.16;
    static constexpr float CAL_S = 241.39;
    static constexpr float CAL_TT = 174.93;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 12.45;
    static constexpr int CAL_S = 11.67;
    static constexpr int CAL_TT = 11.93;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000393";
    constexpr char RET[] = "s100000394";
    constexpr char SUC[] = "s100000397";
    constexpr char LL[] = "s100000398";
    constexpr char ENT_CONDES[] = "s100000418";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000419";
    constexpr char VOLT_S[] = "s100000420";
    constexpr char VOLT_TT[] = "s100000421";

    constexpr char CURR_R[] = "s100000396";
    constexpr char CURR_S[] = "s100000403";
    constexpr char CURR_TT[] = "s100000404";

    constexpr char BAT_INS[] = "xxx";
    constexpr char BAT_RET[] = "xxx";
    constexpr char BAT_SUC[] = "xxx";
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s100000405";
    constexpr char VIBR_Y_SUC[] = "s100000406";
    constexpr char VIBR_Z_SUC[] = "s100000407";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fe:e3"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fe:e7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:97";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:96";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fc:fb";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9904

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9904";
    constexpr char TOPIC[] = "v4/matr0576";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  191.88;
    static constexpr float CAL_S = 10;
    static constexpr float CAL_TT =  10;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.42;
    static constexpr int CAL_S =  10;
    static constexpr int CAL_TT =   10;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000379";
    constexpr char RET[] = "s100000380";
    constexpr char SUC[] = "s100000381";
    constexpr char LL[] = "s100000382";
    constexpr char ENT_CONDES[] = "s100000392";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000383";
    constexpr char VOLT_S[] = "xxxx";
    constexpr char VOLT_TT[] = "xxxx";

    constexpr char CURR_R[] = "s100000384";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s100000388";
    constexpr char BAT_RET[] = "s100000389";
    constexpr char BAT_SUC[] = "s100000390";
    constexpr char BAT_LL[] = "s100000391";

    constexpr char VIBR_X_SUC[] = "s100000385";
    constexpr char VIBR_Y_SUC[] = "s100000386";
    constexpr char VIBR_Z_SUC[] = "s100000387";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:0e"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:9a";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:eb";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:24";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:5e";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9338

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9338";
    constexpr char TOPIC[] = "v4/matr0575";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  223.09;
    static constexpr float CAL_S = 10;
    static constexpr float CAL_TT =  10;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.55;
    static constexpr int CAL_S =  10;
    static constexpr int CAL_TT =   10;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000365";
    constexpr char RET[] = "s100000366";
    constexpr char SUC[] = "s100000367";
    constexpr char LL[] = "s100000368";
    constexpr char ENT_CONDES[] = "s100000378";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000369";
    constexpr char VOLT_S[] = "xxxx";
    constexpr char VOLT_TT[] = "xxxx";

    constexpr char CURR_R[] = "s100000370";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s100000374";
    constexpr char BAT_RET[] = "s100000375";
    constexpr char BAT_SUC[] = "s100000376";
    constexpr char BAT_LL[] = "s100000377";

    constexpr char VIBR_X_SUC[] = "s100000371";
    constexpr char VIBR_Y_SUC[] = "s100000372";
    constexpr char VIBR_Z_SUC[] = "s100000373";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:ff:11"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:e5";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:60";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:e3";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:26:1b";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9436

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9436";
    constexpr char TOPIC[] = "v4/matr0574";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  235.64;
    static constexpr float CAL_S = 167.26;
    static constexpr float CAL_TT =  185.11;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 13.3;
    static constexpr int CAL_S =  12.81;
    static constexpr int CAL_TT =   13.12;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000339";
    constexpr char RET[] = "s100000340";
    constexpr char SUC[] = "s100000345";
    constexpr char LL[] = "s100000346";
    constexpr char ENT_CONDES[] = "s100000364";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000341";
    constexpr char VOLT_S[] = "s100000347";
    constexpr char VOLT_TT[] = "s100000348";

    constexpr char CURR_R[] = "s100000361";
    constexpr char CURR_S[] = "s100000362";
    constexpr char CURR_TT[] = "s100000363";

    constexpr char BAT_INS[] = "s100000357";
    constexpr char BAT_RET[] = "s100000358";
    constexpr char BAT_SUC[] = "s100000359";
    constexpr char BAT_LL[] = "s100000360";

    constexpr char VIBR_X_SUC[] = "s100000354";
    constexpr char VIBR_Y_SUC[] = "s100000355";
    constexpr char VIBR_Z_SUC[] = "s100000356";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:27"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:a4";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:54";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:9e";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:c7";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9494

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9494";
    constexpr char TOPIC[] = "v4/matr0588";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 400;
    static constexpr float CAL_S = 000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91712";
    constexpr char RET[] = "s91713";
    constexpr char SUC[] = "s91714";
    constexpr char LL[] = "s91715";
    constexpr char ENT_CONDES[] = "s91725";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91716";
    constexpr char VOLT_S[] = "xxxxx";
    constexpr char VOLT_TT[] = "xxxxxxx";

    constexpr char CURR_R[] = "s91717";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxxxxx";

    constexpr char BAT_INS[] = "s91721";
    constexpr char BAT_RET[] = "s91722";
    constexpr char BAT_SUC[] = "s91723";
    constexpr char BAT_LL[] = "s91724";

    constexpr char VIBR_X_SUC[] = "s91718";
    constexpr char VIBR_Y_SUC[] = "s91719";
    constexpr char VIBR_Z_SUC[] = "s91720";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:66"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:69";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:05";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:ca";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:c2";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9497

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9497";
    constexpr char TOPIC[] = "v4/matr0590";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 233;
    static constexpr float CAL_S = 000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91740";
    constexpr char RET[] = "s91741";
    constexpr char SUC[] = "s91742";
    constexpr char LL[] = "s91743";
    constexpr char ENT_CONDES[] = "s91753";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91744";
    constexpr char VOLT_S[] = "xxxxx";
    constexpr char VOLT_TT[] = "xxxxxxx";

    constexpr char CURR_R[] = "s91745";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxxxxx";

    constexpr char BAT_INS[] = "s91749";
    constexpr char BAT_RET[] = "s91750";
    constexpr char BAT_SUC[] = "s91751";
    constexpr char BAT_LL[] = "s91752";

    constexpr char VIBR_X_SUC[] = "s91746";
    constexpr char VIBR_Y_SUC[] = "s91747";
    constexpr char VIBR_Z_SUC[] = "s91748";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:fa"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2f:03";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:29";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:16";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:c2";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9498

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9498";
    constexpr char TOPIC[] = "v4/matr0589";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 295.39;
    static constexpr float CAL_S = 000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.67;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91726";
    constexpr char RET[] = "s91727";
    constexpr char SUC[] = "s91728";
    constexpr char LL[] = "s91729";
    constexpr char ENT_CONDES[] = "s91739";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91730";
    constexpr char VOLT_S[] = "xxxxx";
    constexpr char VOLT_TT[] = "xxxxxxx";

    constexpr char CURR_R[] = "s91731";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxxxxx";

    constexpr char BAT_INS[] = "s91735";
    constexpr char BAT_RET[] = "s91736";
    constexpr char BAT_SUC[] = "s91737";
    constexpr char BAT_LL[] = "s91738";

    constexpr char VIBR_X_SUC[] = "s91732";
    constexpr char VIBR_Y_SUC[] = "s91733";
    constexpr char VIBR_Z_SUC[] = "s91734";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:ce"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:25:e8";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:3a";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:bc";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:c2";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9978

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9978";
    constexpr char TOPIC[] = "v4/matr0572";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  100.34;
    static constexpr float CAL_S = 229.81;
    static constexpr float CAL_TT =  135.08;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT =  14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000313";
    constexpr char RET[] = "s100000314";
    constexpr char SUC[] = "s100000317";
    constexpr char LL[] = "s100000318";
    constexpr char ENT_CONDES[] = "s100000338";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000315";
    constexpr char VOLT_S[] = "s100000321";
    constexpr char VOLT_TT[] = "s100000322";

    constexpr char CURR_R[] = "s100000316";
    constexpr char CURR_S[] = "s100000323";
    constexpr char CURR_TT[] = "s100000324";

    constexpr char BAT_INS[] = "s100000331";
    constexpr char BAT_RET[] = "s100000332";
    constexpr char BAT_SUC[] = "s100000333";
    constexpr char BAT_LL[] = "s100000334";

    constexpr char VIBR_X_SUC[] = "s100000325";
    constexpr char VIBR_Y_SUC[] = "s100000326";
    constexpr char VIBR_Z_SUC[] = "s100000327";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fc:eb"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fc:fa";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:84";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:ff:83";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:0c";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9980

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9980";
    constexpr char TOPIC[] = "v4/matr0570";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  79.02;
    static constexpr float CAL_S = 71.36;
    static constexpr float CAL_TT =  69.67;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.49;
    static constexpr int CAL_S = 17.5;
    static constexpr int CAL_TT =  16.55;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000261";
    constexpr char RET[] = "s100000262";
    constexpr char SUC[] = "s100000265";
    constexpr char LL[] = "s100000266";
    constexpr char ENT_CONDES[] = "s100000286";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000263";
    constexpr char VOLT_S[] = "s100000269";
    constexpr char VOLT_TT[] = "s100000270";

    constexpr char CURR_R[] = "s100000264";
    constexpr char CURR_S[] = "s100000271";
    constexpr char CURR_TT[] = "s100000272";

    constexpr char BAT_INS[] = "s100000279";
    constexpr char BAT_RET[] = "s100000280";
    constexpr char BAT_SUC[] = "s100000281";
    constexpr char BAT_LL[] = "s100000282";

    constexpr char VIBR_X_SUC[] = "s100000273";
    constexpr char VIBR_Y_SUC[] = "s100000274";
    constexpr char VIBR_Z_SUC[] = "s100000275";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:07"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:0b";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:f4";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:f7";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:0c";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9980 CKT 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9980";
    constexpr char TOPIC[] = "v4/matr0570";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  79.02;
    static constexpr float CAL_S = 71.36;
    static constexpr float CAL_TT =  69.67;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R =  17.31;
    static constexpr int CAL_S = 17.53;
    static constexpr int CAL_TT =  17.92;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000261";
    constexpr char RET[] = "s100000262";
    constexpr char SUC[] = "s100000267";
    constexpr char LL[] = "s100000268";
    constexpr char ENT_CONDES[] = "s100000286";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "xxxxxxxxxxxx";
    constexpr char VOLT_S[] = "xxxxxxxxxxxxxxx";
    constexpr char VOLT_TT[] = "xxxxxxxxxxxxxx";

    constexpr char CURR_R[] = "s100000283";
    constexpr char CURR_S[] = "s100000284";
    constexpr char CURR_TT[] = "s100000285";

    constexpr char BAT_INS[] = "xxxxx";
    constexpr char BAT_RET[] = "xx";
    constexpr char BAT_SUC[] = "x";
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s100000276";
    constexpr char VIBR_Y_SUC[] = "s100000277";
    constexpr char VIBR_Z_SUC[] = "s100000278";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:XX:XX"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:XX:XX";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:ff:81";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:f3";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:XX:XX";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9996

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9996";
    constexpr char TOPIC[] = "v4/matr0595";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 520.88;
    static constexpr float CAL_S = 452.75;
    static constexpr float CAL_TT = 462.42;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 4.98;
    static constexpr int CAL_S = 4.88;
    static constexpr int CAL_TT = 4.50;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91818";
    constexpr char RET[] = "s91819";
    constexpr char SUC[] = "s91820";
    constexpr char LL[] = "s91821";
    constexpr char ENT_CONDES[] = "s91831";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91822";
    constexpr char VOLT_S[] = "xxxx";
    constexpr char VOLT_TT[] = "xxxxx";

    constexpr char CURR_R[] = "s91823";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxx";

    constexpr char BAT_INS[] = "s91827";
    constexpr char BAT_RET[] = "s91828";
    constexpr char BAT_SUC[] = "s91829";
    constexpr char BAT_LL[] = "s91830";

    constexpr char VIBR_X_SUC[] = "s91824";
    constexpr char VIBR_Y_SUC[] = "s91825";
    constexpr char VIBR_Z_SUC[] = "s91826";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:xx:xx"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:xx:xx";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:xx:xx";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:xx:xx";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:41";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

9605

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9605";
    constexpr char TOPIC[] = "v4/matr0526";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 308.03;
    static constexpr float CAL_S = 100.51;
    static constexpr float CAL_TT = 162.54;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 16.21;
    static constexpr int CAL_TT = 12.44;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s21093";
    constexpr char RET[] = "s21094";
    constexpr char SUC[] = "s21095";
    constexpr char LL[] = "s21096";
    constexpr char ENT_CONDES[] = "xxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s21097";
    constexpr char VOLT_S[] = "s21106";
    constexpr char VOLT_TT[] = "s21107";

    constexpr char CURR_R[] = "s21098";
    constexpr char CURR_S[] = "s21108";
    constexpr char CURR_TT[] = "s21109";

    constexpr char BAT_INS[] = "s21102";
    constexpr char BAT_RET[] = "s21103";
    constexpr char BAT_SUC[] = "s21104";
    constexpr char BAT_LL[] = "s21105";

    constexpr char VIBR_X_SUC[] = "s21099";
    constexpr char VIBR_Y_SUC[] = "s21100";
    constexpr char VIBR_Z_SUC[] = "s21101";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:da"; //Ble2
    constexpr char TEMP_RET[] = "";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:dd";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:6d";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};

#endif
```

9321

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "9321";
    constexpr char TOPIC[] = "v4/matr0586";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  103.78;
    static constexpr float CAL_S = 106.34;
    static constexpr float CAL_TT = 33.45;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.28;
    static constexpr int CAL_S = 16.99;
    static constexpr int CAL_TT = 16.49;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91666";
    constexpr char RET[] = "s91667";
    constexpr char SUC[] = "s91668";
    constexpr char LL[] = "s91669";
    constexpr char ENT_CONDES[] = "s91683";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91670";
    constexpr char VOLT_S[] = "s91671";
    constexpr char VOLT_TT[] = "s91672";

    constexpr char CURR_R[] = "s91673";
    constexpr char CURR_S[] = "s91674";
    constexpr char CURR_TT[] = "s91675";

    constexpr char BAT_INS[] = "s91679";
    constexpr char BAT_RET[] = "s91680";
    constexpr char BAT_SUC[] = "s91681";
    constexpr char BAT_LL[] = "s91682";

    constexpr char VIBR_X_SUC[] = "s91676";
    constexpr char VIBR_Y_SUC[] = "s91677";
    constexpr char VIBR_Z_SUC[] = "s91678";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:32"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:26:0d";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:2e";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:37";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:26:1f";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10052 CKT 1

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10052";
    constexpr char TOPIC[] = "v4/matr0571";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  489.37;
    static constexpr float CAL_S =  107.04;
    static constexpr float CAL_TT =  110.62;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT =  14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s100000287";
    constexpr char RET[] = "s100000288";
    constexpr char SUC[] = "s100000291";
    constexpr char LL[] = "s100000292";
    constexpr char ENT_CONDES[] = "s100000312";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s100000289";
    constexpr char VOLT_S[] = "s100000295";
    constexpr char VOLT_TT[] = "s100000296";

    constexpr char CURR_R[] = "s100000290";
    constexpr char CURR_S[] = "s100000297";
    constexpr char CURR_TT[] = "s100000298";

    constexpr char BAT_INS[] = "s100000305";
    constexpr char BAT_RET[] = "s100000306";
    constexpr char BAT_SUC[] = "s100000307";
    constexpr char BAT_LL[] = "s100000308";

    constexpr char VIBR_X_SUC[] = "s100000299";
    constexpr char VIBR_Y_SUC[] = "s100000300";
    constexpr char VIBR_Z_SUC[] = "s100000301";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:0d"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:bd";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:f2";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:dd";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:23";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10052 CKT 2

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac_Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10052";
    constexpr char TOPIC[] = "v4/matr0571b";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R =  489.37;
    static constexpr float CAL_S =  107.04;
    static constexpr float CAL_TT =  110.62;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14;
    static constexpr int CAL_S = 14;
    static constexpr int CAL_TT =  14;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 32;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 33;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 35;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "xxx";
    constexpr char RET[] = "xxx";
    constexpr char SUC[] = "xxx";
    constexpr char LL[] = "s100000293";
    constexpr char ENT_CONDES[] = "s100000294";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "xxx";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s100000309";
    constexpr char CURR_S[] = "s100000310";
    constexpr char CURR_TT[] = "ss100000311";

    constexpr char BAT_INS[] = "xxx";
    constexpr char BAT_RET[] = "xxx";
    constexpr char BAT_SUC[] = "xxx";
    constexpr char BAT_LL[] = "xxx";

    constexpr char VIBR_X_SUC[] = "s100000302";
    constexpr char VIBR_Y_SUC[] = "s100000303";
    constexpr char VIBR_Z_SUC[] = "s100000304";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:13:fd:0d"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:13:fd:bd";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:13:fe:e4";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:13:fe:d9";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:13:fd:23";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

12826

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "12826";
    constexpr char TOPIC[] = "v4/matr0591";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 107.19;
    static constexpr float CAL_S = 108.02;
    static constexpr float CAL_TT = 32.95;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.39;
    static constexpr int CAL_S = 17.04;
    static constexpr int CAL_TT = 17.06;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91754";
    constexpr char RET[] = "s91755";
    constexpr char SUC[] = "s91756";
    constexpr char LL[] = "s91757";
    constexpr char ENT_CONDES[] = "s91771";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91758";
    constexpr char VOLT_S[] = "s91759";
    constexpr char VOLT_TT[] = "s91760";

    constexpr char CURR_R[] = "s91761";
    constexpr char CURR_S[] = "s91762";
    constexpr char CURR_TT[] = "s91763";

    constexpr char BAT_INS[] = "s91767";
    constexpr char BAT_RET[] = "s91768";
    constexpr char BAT_SUC[] = "s91769";
    constexpr char BAT_LL[] = "s91770";

    constexpr char VIBR_X_SUC[] = "s91764";
    constexpr char VIBR_Y_SUC[] = "s91765";
    constexpr char VIBR_Z_SUC[] = "s91766";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:d0"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:d3";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:73";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:2f";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:68";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

12828

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "12828";
    constexpr char TOPIC[] = "v4/matr0592";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 111;
    static constexpr float CAL_S =  109.77;
    static constexpr float CAL_TT =  31.63;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R =  16.75 ;
    static constexpr int CAL_S = 17.40 ;
    static constexpr int CAL_TT =  16.21;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 39;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 35;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 32;
    static constexpr int VOLTAGE_S = 34;
```

```
    static constexpr int VOLTAGE_TT = 36;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91772";
    constexpr char RET[] = "s91773";
    constexpr char SUC[] = "s91774";
    constexpr char LL[] = "s91775";
    constexpr char ENT_CONDES[] = "s91789";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91776";
    constexpr char VOLT_S[] = "s91777";
    constexpr char VOLT_TT[] = "s91778";

    constexpr char CURR_R[] = "s91779";
    constexpr char CURR_S[] = "s91780";
    constexpr char CURR_TT[] = "s91781";

    constexpr char BAT_INS[] = "s91785";
    constexpr char BAT_RET[] = "s91786";
    constexpr char BAT_SUC[] = "s91787";
    constexpr char BAT_LL[] = "s91788";

    constexpr char VIBR_X_SUC[] = "s91782";
    constexpr char VIBR_Y_SUC[] = "s91783";
    constexpr char VIBR_Z_SUC[] = "s91784";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:e6"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:34";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:39";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:70";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:68";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

12862

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "12862";
    constexpr char TOPIC[] = "v4/matr0580";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 162.75;
    static constexpr float CAL_S = 000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.42;
    static constexpr int CAL_S = 000;
    static constexpr int CAL_TT = 000;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 36;
    static constexpr int CURRENT_TT = 39;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 33;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91555";
    constexpr char RET[] = "s91556";
    constexpr char SUC[] = "s91557";
    constexpr char LL[] = "s91558";
    constexpr char ENT_CONDES[] = "s91568";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91559";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91560";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxx";

    constexpr char BAT_INS[] = "s91564";
    constexpr char BAT_RET[] = "s91565";
    constexpr char BAT_SUC[] = "s91566";
    constexpr char BAT_LL[] = "s91567";

    constexpr char VIBR_X_SUC[] = "s91561";
    constexpr char VIBR_Y_SUC[] = "s91562";
    constexpr char VIBR_Z_SUC[] = "s91563";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:fd"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:19:d4";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:84";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:26:37";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:26:1f";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

13217

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "13217";
    constexpr char TOPIC[] = "v4/matr0542";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 266;
    static constexpr float CAL_S = 452.75;
    static constexpr float CAL_TT = 462.42;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 12.76;
    static constexpr int CAL_S = 4.88;
    static constexpr int CAL_TT = 4.50;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 39;
```

```
    static constexpr int VOLTAGE_TT = 32;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91406";
    constexpr char RET[] = "s91407";
    constexpr char SUC[] = "s91408";
    constexpr char LL[] = "s91409";
    constexpr char ENT_CONDES[] = "s91419";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91411";
    constexpr char VOLT_S[] = "xxxxx";
    constexpr char VOLT_TT[] = "xxxx";

    constexpr char CURR_R[] = "s91410";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxxx";

    constexpr char BAT_INS[] = "s91415";
    constexpr char BAT_RET[] = "s91416";
    constexpr char BAT_SUC[] = "s91417";
    constexpr char BAT_LL[] = "s91418";

    constexpr char VIBR_X_SUC[] = "s91412";
    constexpr char VIBR_Y_SUC[] = "s91413";
    constexpr char VIBR_Z_SUC[] = "s91414";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:e9"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:b2";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:26:02";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:63";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "xx:xx:29:0e:19:xx";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

33213

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "33213";
    constexpr char TOPIC[] = "v4/matr0579";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 257.13;
    static constexpr float CAL_S = 452.75;
    static constexpr float CAL_TT = 462.42;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 17.04;
    static constexpr int CAL_S = 4.88;
    static constexpr int CAL_TT = 4.50;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 39;
```

```
    static constexpr int VOLTAGE_TT = 32;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91541";
    constexpr char RET[] = "s91542";
    constexpr char SUC[] = "s91543";
    constexpr char LL[] = "s91544";
    constexpr char ENT_CONDES[] = "s91554";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91545";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91546";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91550";
    constexpr char BAT_RET[] = "s91551";
    constexpr char BAT_SUC[] = "s91552";
    constexpr char BAT_LL[] = "s91553";

    constexpr char VIBR_X_SUC[] = "s91547";
    constexpr char VIBR_Y_SUC[] = "s91548";
    constexpr char VIBR_Z_SUC[] = "s91549";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:2e:d0"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:25:e7";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:25:f0";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:28";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:2e:ec";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

36517

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "36517";
    constexpr char TOPIC[] = "v4/matr0543";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 206;
    static constexpr float CAL_S = 0000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.6;
    static constexpr int CAL_S = 000;
    static constexpr int CAL_TT = 000;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91420";
    constexpr char RET[] = "s91421";
    constexpr char SUC[] = "s91422";
    constexpr char LL[] = "s91423";
    constexpr char ENT_CONDES[] = "xxxx";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91424";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxxx";

    constexpr char CURR_R[] = "s91425";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91429";
    constexpr char BAT_RET[] = "s91430";
    constexpr char BAT_SUC[] = "s91431";
    constexpr char BAT_LL[] = "s91432";

    constexpr char VIBR_X_SUC[] = "s91426";
    constexpr char VIBR_Y_SUC[] = "s91427";
    constexpr char VIBR_Z_SUC[] = "s91428";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:25:e5"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:b6";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:ea";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:2f:a3";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "xx:xx:29:0e:19:7a";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

36518

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
```

```
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "36518";
    constexpr char TOPIC[] = "v4/matr0582";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 161.04;
    static constexpr float CAL_S = 0000;
    static constexpr float CAL_TT = 000;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 15.36;
    static constexpr int CAL_S = 000;
    static constexpr int CAL_TT = 000;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 36;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 32;
```

```
    static constexpr int VOLTAGE_TT = 39;
};

// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91583";
    constexpr char RET[] = "s91584";
    constexpr char SUC[] = "s91585";
    constexpr char LL[] = "s91586";
    constexpr char ENT_CONDES[] = "s91596";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91587";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxxx";

    constexpr char CURR_R[] = "s91588";
    constexpr char CURR_S[] = "xxxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91592";
    constexpr char BAT_RET[] = "s91593";
    constexpr char BAT_SUC[] = "s91594";
    constexpr char BAT_LL[] = "s91595";

    constexpr char VIBR_X_SUC[] = "s91589";
    constexpr char VIBR_Y_SUC[] = "s91590";
    constexpr char VIBR_Z_SUC[] = "s91591";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:26:1e"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:26:0a";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:92";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:25:de";   //Ble3
```

```
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:41";   //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";   //Ble4
};

#endif
```

10017

```
#ifndef _ENV_H
#define _ENV_H

#include <stdint.h>

// Definir níveis de log
#define LOG_LEVEL_NONE  0  // Sem logs
#define LOG_LEVEL_ERROR 1  // Apenas erros
#define LOG_LEVEL_WARN  2  // Avisos e erros
#define LOG_LEVEL_INFO  3  // Informações, avisos e erros
#define LOG_LEVEL_DEBUG 4  // Mensagens de debug, informações, avisos e erros

// Definir qual o nível de log ativo
#define CURRENT_LOG_LEVEL LOG_LEVEL_ERROR

// Funções de log condicional com base no nível
#define LOG_LEVEL_ERROR 1
#define LOG_LEVEL_WARN  2
#define LOG_LEVEL_INFO  3
#define LOG_LEVEL_DEBUG 4

// Defina o nível de log atual
#define CURRENT_LOG_LEVEL LOG_LEVEL_DEBUG

// Macros de log condicional com suporte para mensagens simples e formatadas
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_ERROR
#define LOG_ERROR(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[ERROR] " fmt "\n", ##__VA_ARGS__); \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_ERROR(fmt, ...)
#endif
```

```
#if CURRENT_LOG_LEVEL >= LOG_LEVEL_WARN
#define LOG_WARN(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[WARN] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_WARN(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_INFO
#define LOG_INFO(fmt, ...)                                  \
    do                                                      \
    {                                                       \
        Serial.printf("[INFO] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                     \
    } while (0)
#else
#define LOG_INFO(fmt, ...)
#endif

#if CURRENT_LOG_LEVEL >= LOG_LEVEL_DEBUG
#define LOG_DEBUG(fmt, ...)                                  \
    do                                                       \
    {                                                        \
        Serial.printf("[DEBUG] " fmt "\n", ##__VA_ARGS__);  \
        Serial.flush();                                      \
    } while (0)
#else
#define LOG_DEBUG(fmt, ...)
#endif

// WiFi Network Credentials
constexpr char NETWORK_CLIENT[] = "SmartVac Telemetria";
constexpr char PASSW[] = "br@skem#2023";

constexpr uint32_t DELAY_MS_READER_TASK = 300; //300ms para que os envios sejam feitos a cada
20s
// Define o atraso em milissegundos para tarefas de leitura (10.000 parece ser um número
```

```
mágico)

// Equipment Identifiers
namespace Equipment
{
    constexpr char TAG[] = "10017";
    constexpr char TOPIC[] = "v4/matr0587";
    constexpr char SENDING_SERVER[] = "web.smartvac.app";
    constexpr uint16_t SENDING_PORT = 1883;
    constexpr unsigned long SENDING_VELOCITY = 115200;
    constexpr uint16_t SENDING_KEEPALIVE = 200;
    constexpr uint32_t DELAY_MS = 500; // Delay time in milliseconds for tasks
}

// Voltage Calibration Constants
struct VoltageCalibration
{
    static constexpr float CAL_R = 223.10;
    static constexpr float CAL_S = 0;
    static constexpr float CAL_TT = 0;
};

// Current Calibration Constants
struct CurrentCalibration
{
    static constexpr int CAL_R = 14.5;
    static constexpr int CAL_S = 0;
    static constexpr int CAL_TT = 0;
};

// Pin Definitions
struct Pins
{
    static constexpr int CURRENT_R = 34;
    static constexpr int CURRENT_S = 33;
    static constexpr int CURRENT_TT = 32;
    static constexpr int TEMPERATURE = 16;
    static constexpr int VOLTAGE_R = 35;
    static constexpr int VOLTAGE_S = 36;
    static constexpr int VOLTAGE_TT = 39;
};
```

```
// Sensor Addresses
namespace Sensors
{
    constexpr char INS[] = "s91698";
    constexpr char RET[] = "s91699";
    constexpr char SUC[] = "s91700";
    constexpr char LL[] = "s91701";
    constexpr char ENT_CONDES[] = "s91711";
    constexpr char SAD_CONDES[] = "xxxxx";// essa temp não ta sendo enviada, ignorar ela ou
adicionar nos envios se necessario

    constexpr char VOLT_R[] = "s91702";
    constexpr char VOLT_S[] = "xxx";
    constexpr char VOLT_TT[] = "xxx";

    constexpr char CURR_R[] = "s91703";
    constexpr char CURR_S[] = "xxx";
    constexpr char CURR_TT[] = "xxx";

    constexpr char BAT_INS[] = "s91707";
    constexpr char BAT_RET[] = "s91708";
    constexpr char BAT_SUC[] = "s91709";
    constexpr char BAT_LL[] = "s91710";

    constexpr char VIBR_X_SUC[] = "s91704";
    constexpr char VIBR_Y_SUC[] = "s91705";
    constexpr char VIBR_Z_SUC[] = "s91706";

}

// BLE Sensor MAC Addresses
namespace BLEAddresses
{
    constexpr char TEMP_INSU[] = "bc:57:29:0e:19:35"; //Ble2
    constexpr char TEMP_RET[] = "bc:57:29:0e:2e:ee";  //Ble1
    constexpr char TEMP_SUC[] = "bc:57:29:0e:19:b4";  //Ble4
    constexpr char TEMP_LL[] = "bc:57:29:0e:19:3a";   //Ble3
    constexpr char TEMP_ENT_CONDES[] = "bc:57:29:0e:19:81";  //Ble4
    constexpr char TEMP_SAI_CONDES[] = "";  //Ble4
};
```

```
#endif
```