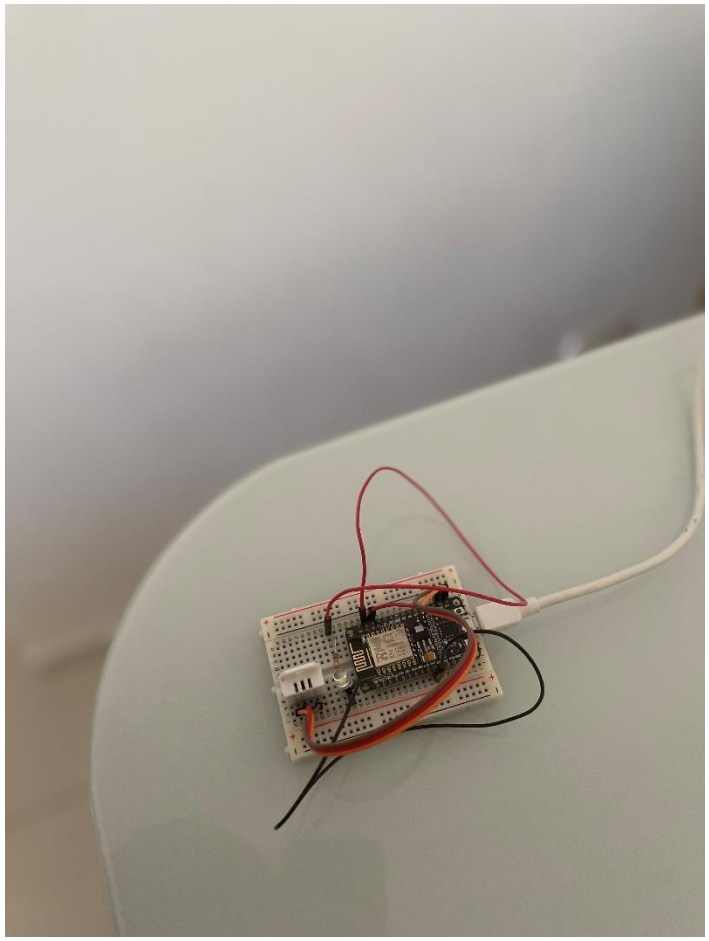
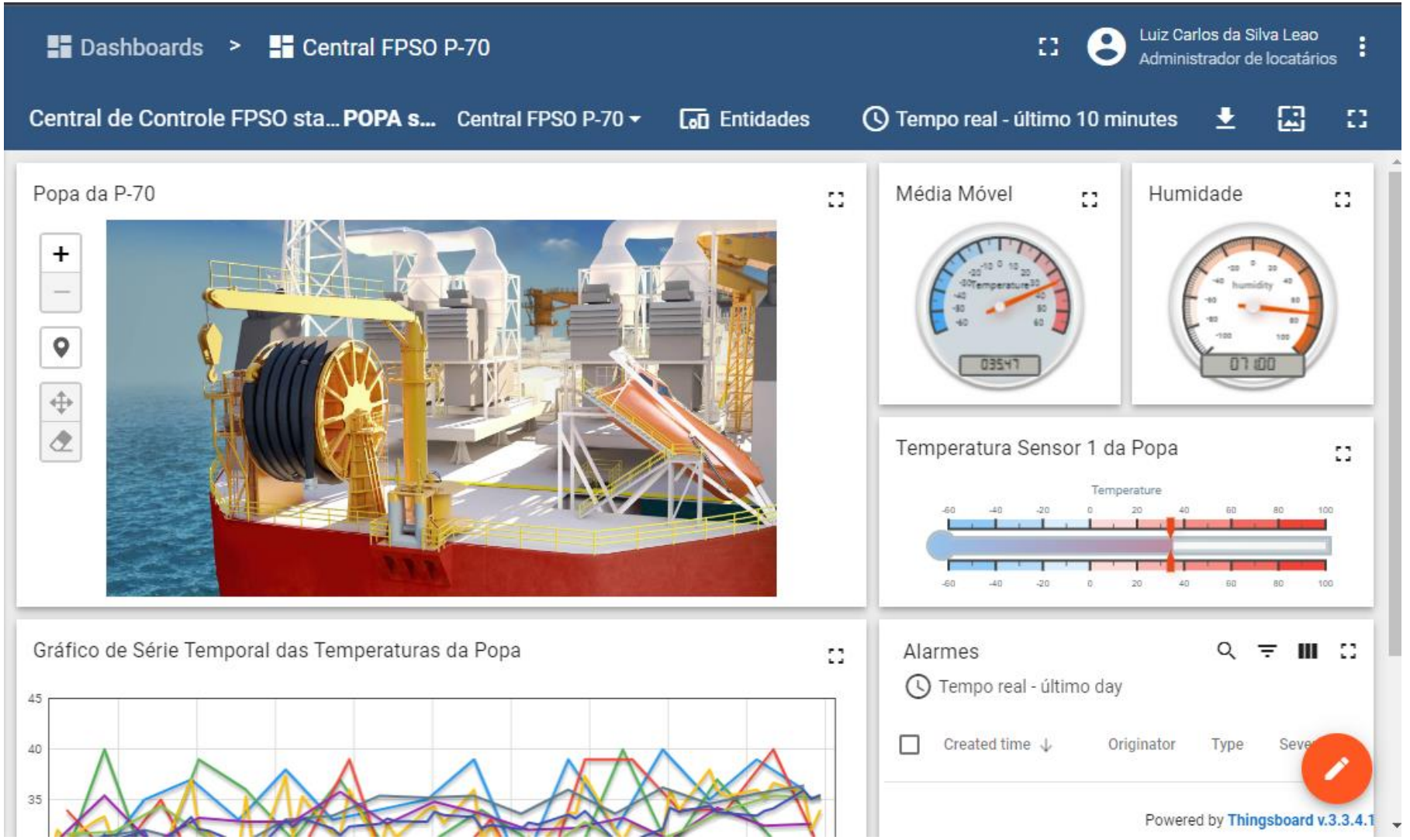
Resultados do Trabalho Prático no ThingsBoard do aluno Luiz Leão – Data de entrega: 08/07/2022

a) Foto do dispositivo NodeMCU + o sensor DHT 22 + o LED para acendimento via RPC:

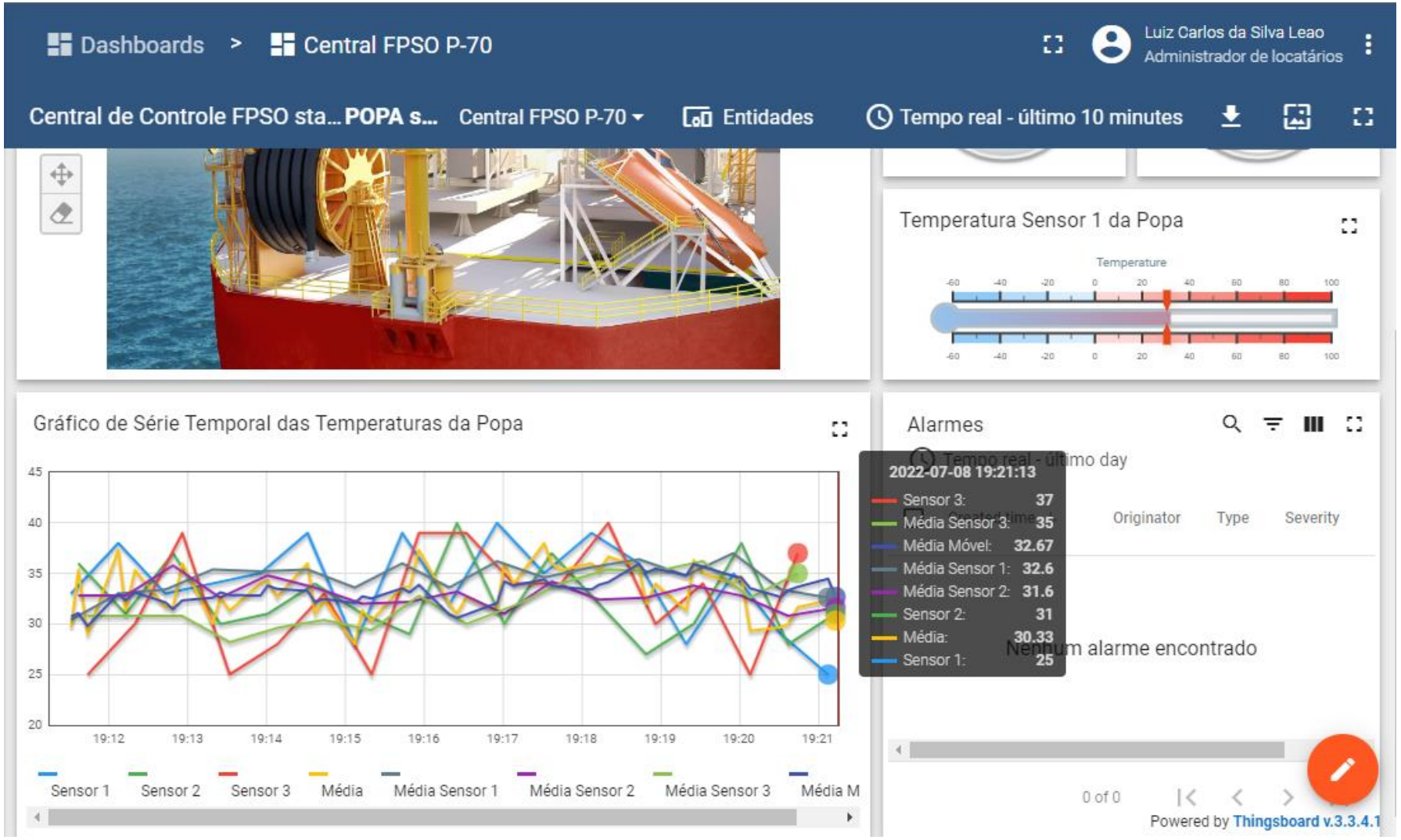


b) Dashboard – Painel principal (Áreas da FPSO P-70: Popa, Passadiço, Casa de Máquinas, Convés e a Proa – cada uma com 3 dispositivos)

c) Dashboard – Popa 1 (com a apresentação das médias e média móvel)

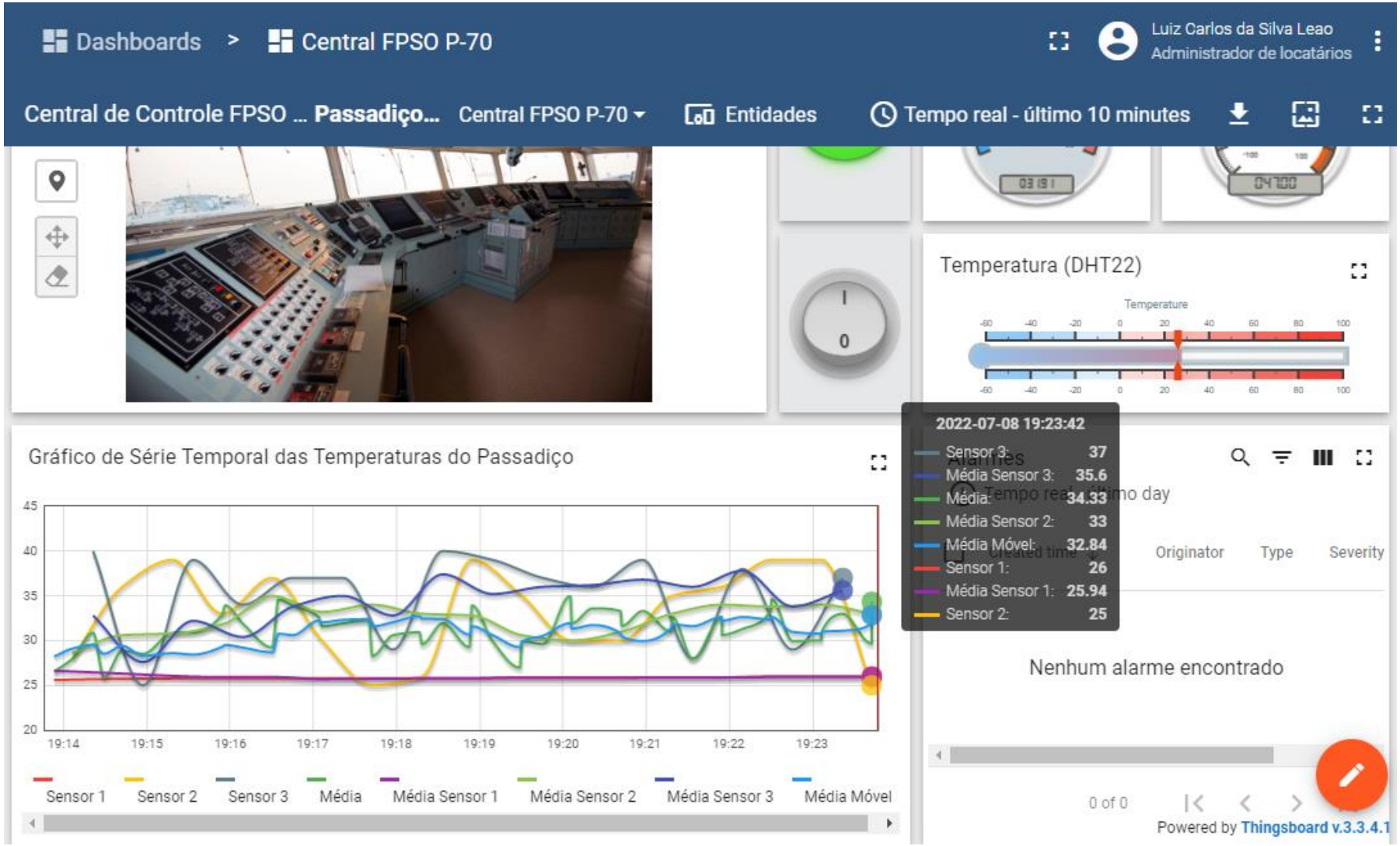


d) Dashboard Popa

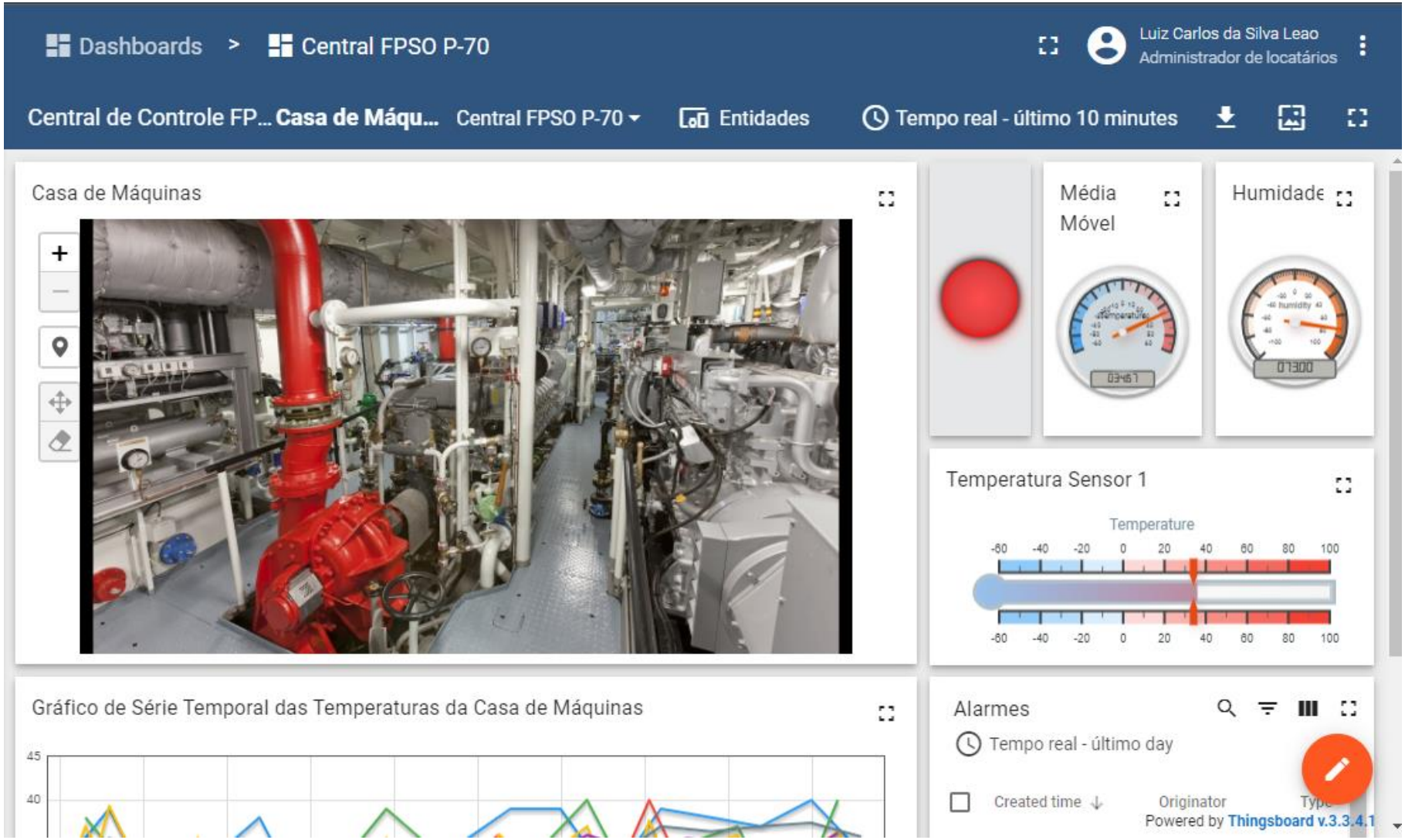


e) Dashboard – Passadiço (com o atuador que liga e desliga via RPC o LED do dispositivo)

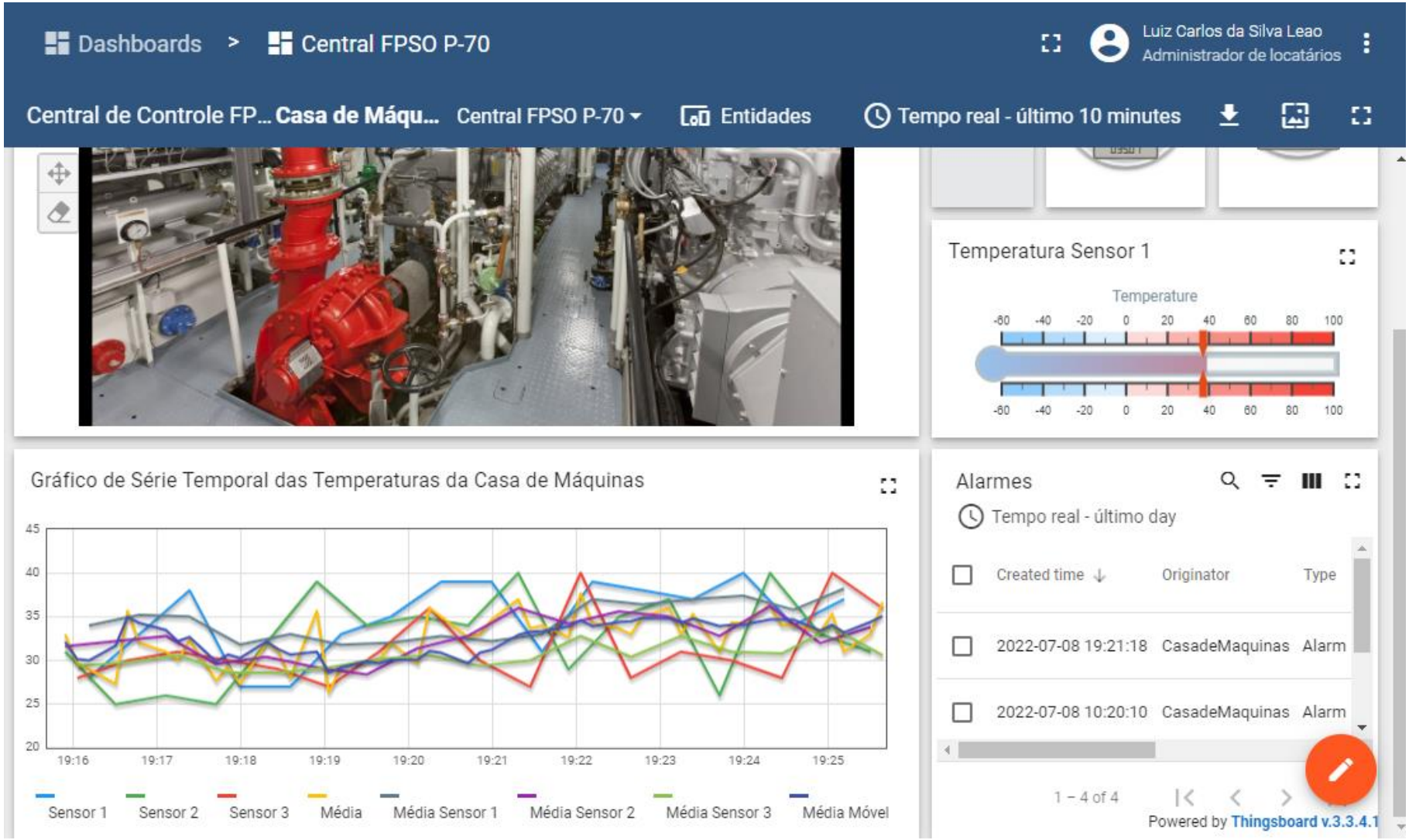
f) Dashboard – Passadiço (continuação):



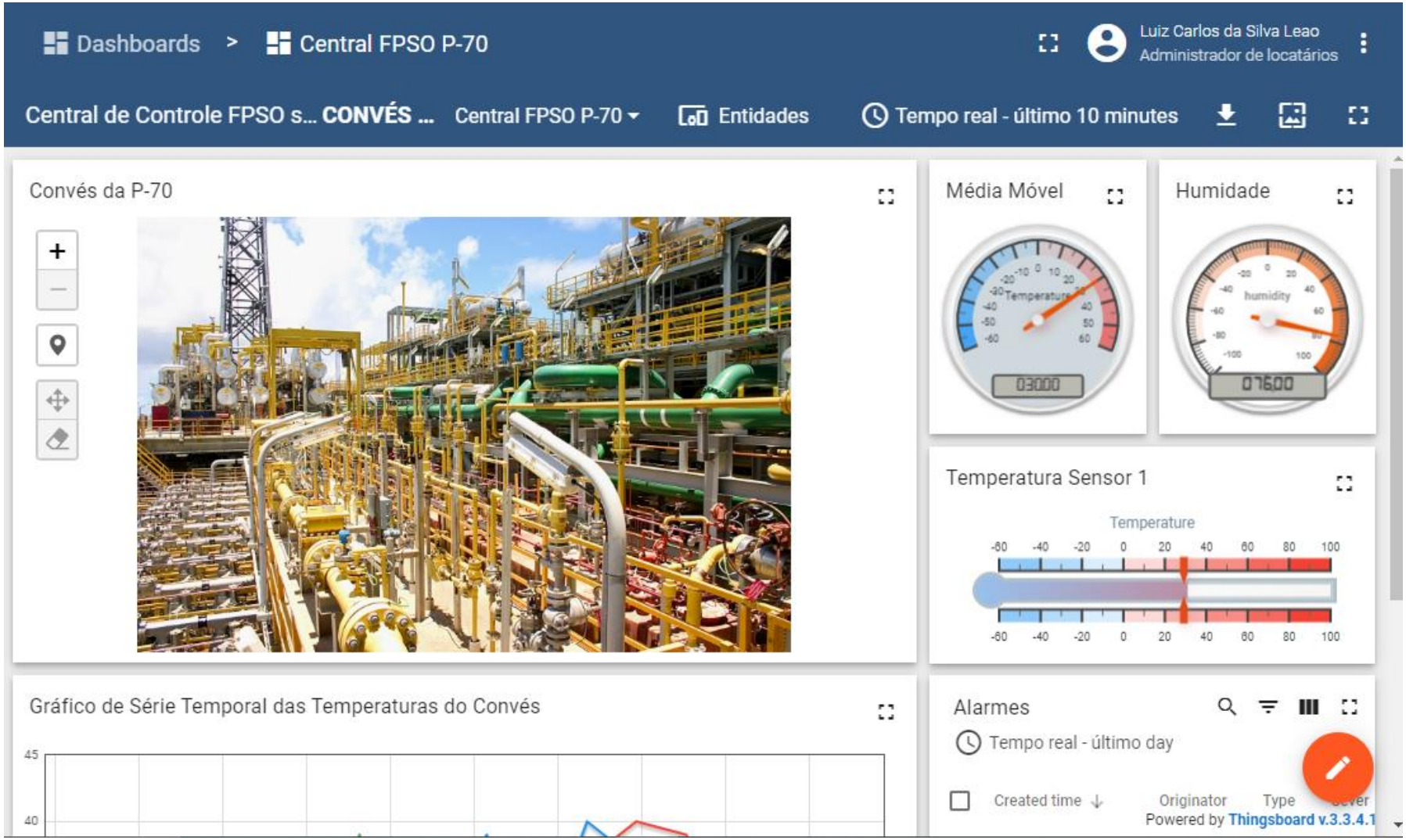
g) Dashboard – Casa de Máquinas (com o LED que acende e apaga conforme o alarme da média móvel é disparado)



h) Continuação Dashboard Casa de Máquinas (com a indicativo dos alarmes gerados):

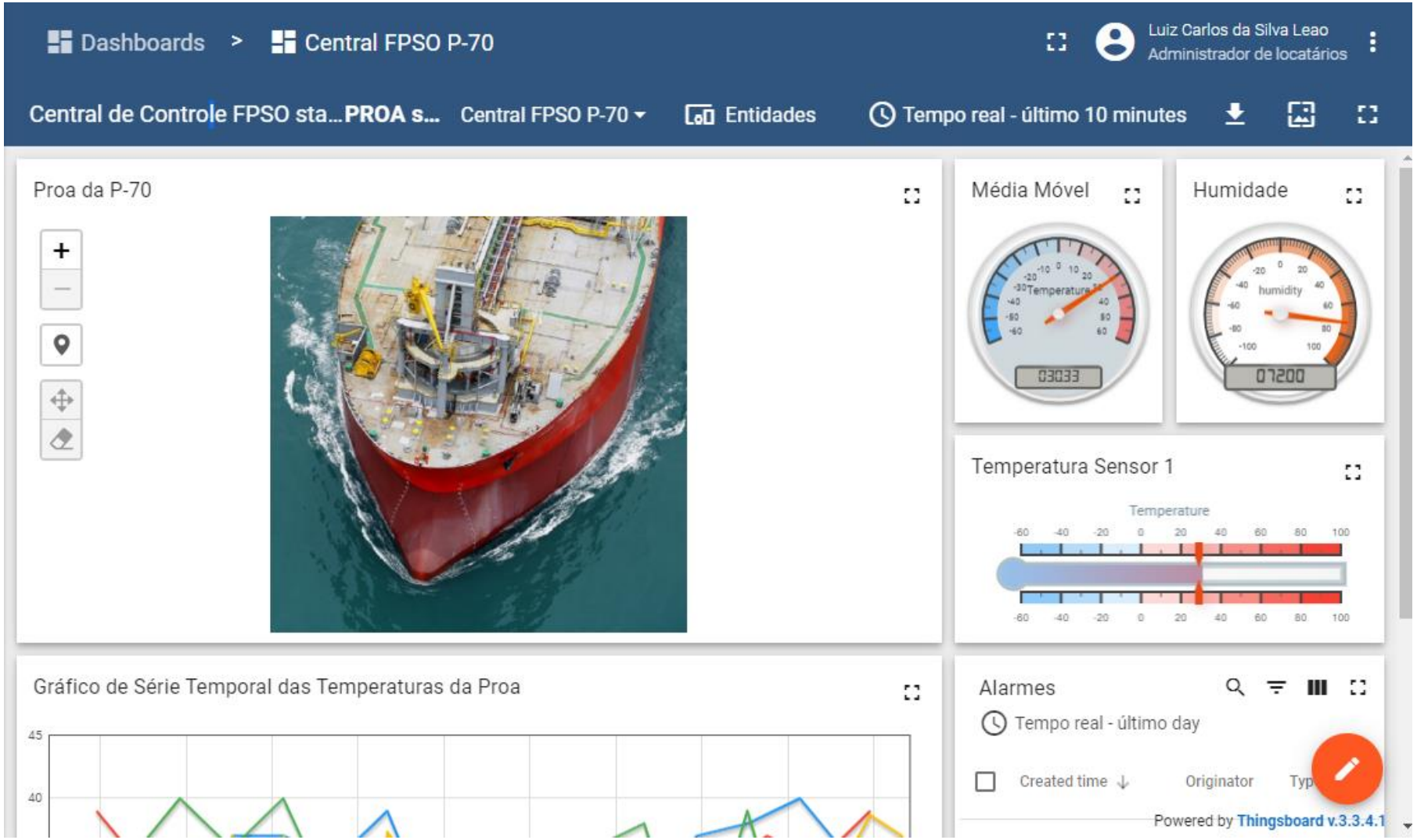


i) Dashboard Convés:

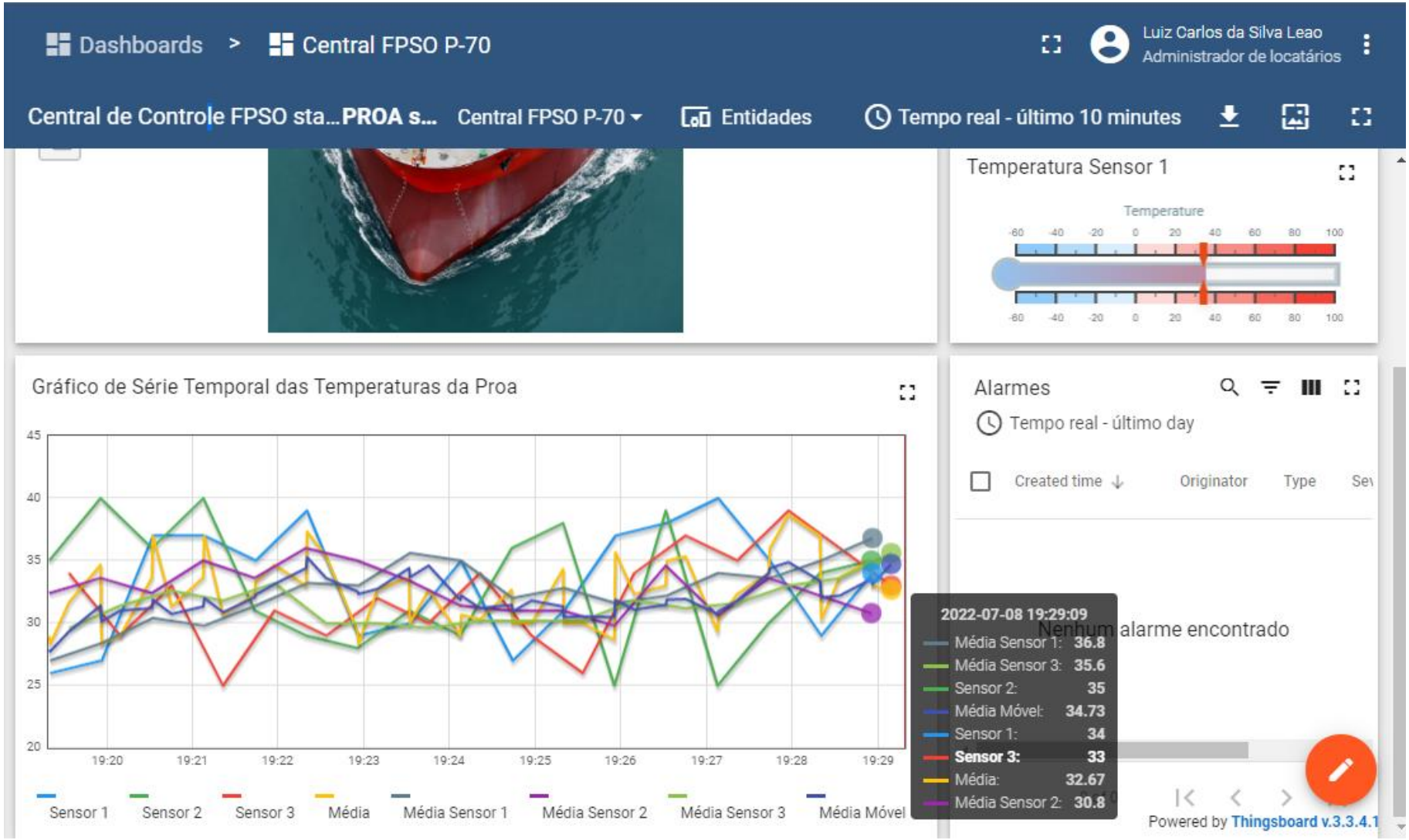


j) Dashboard Convés (continuação):

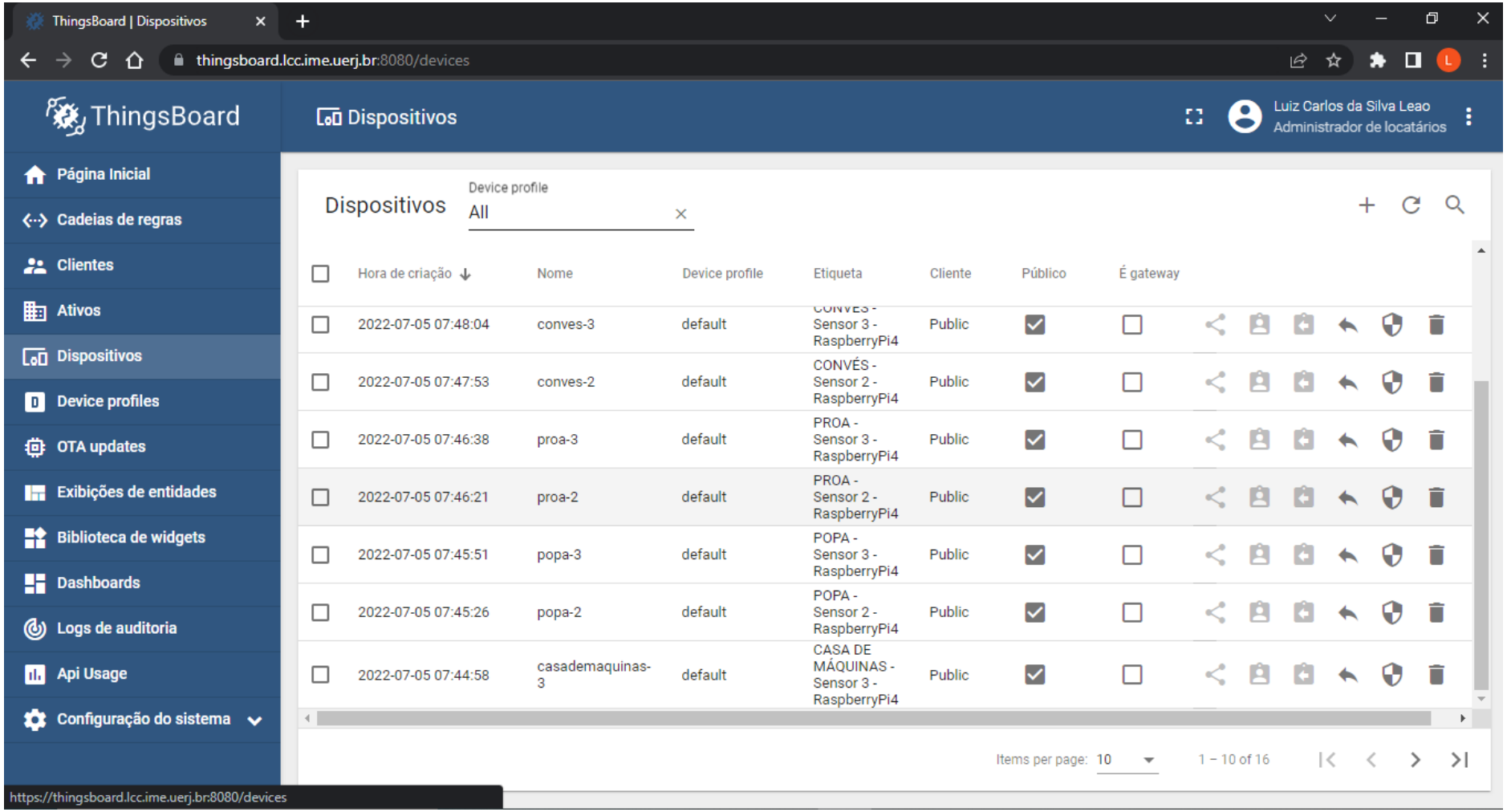
k) Dashboard da Proa:



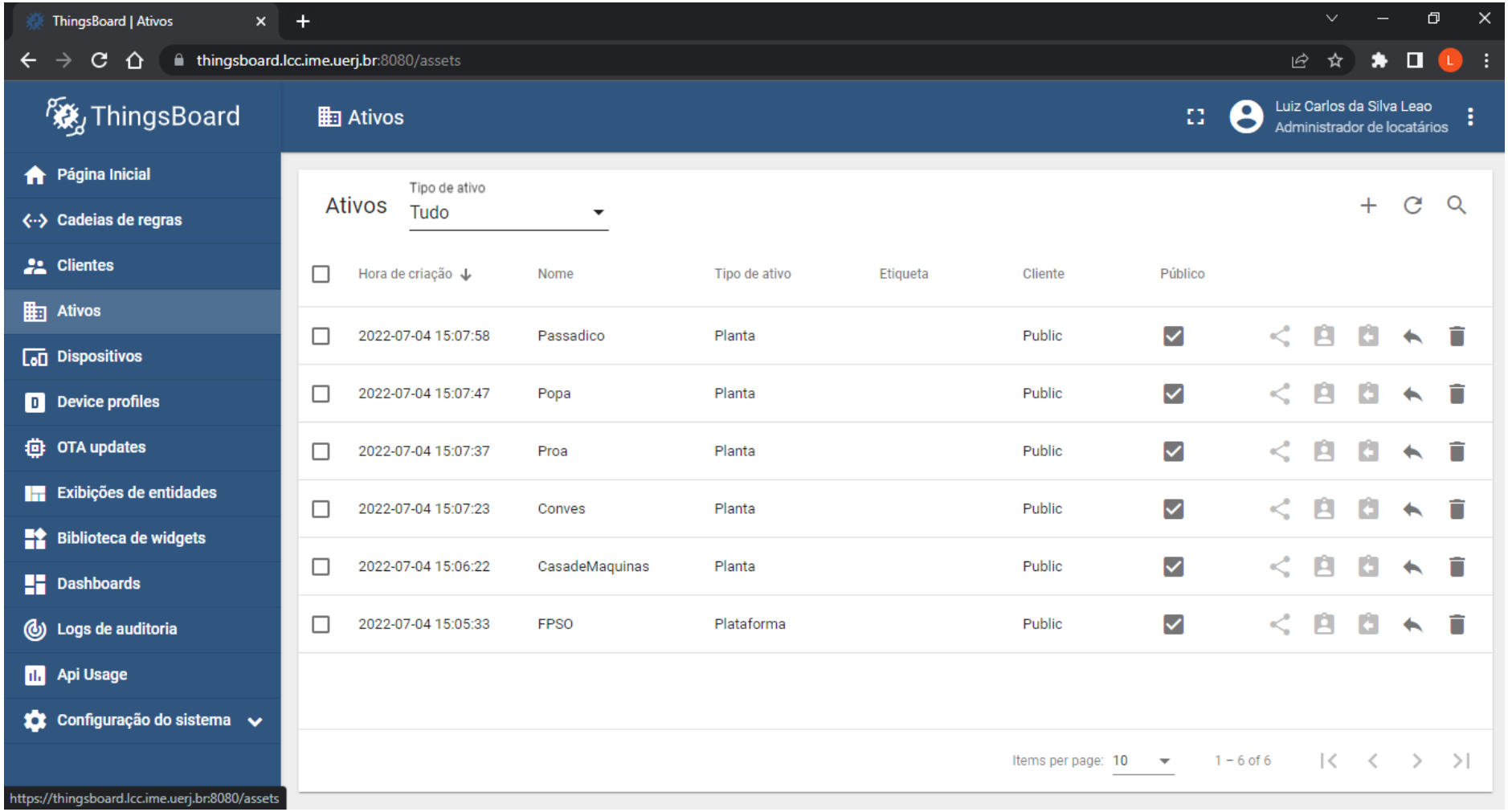
I) Dashboard da Proa (continuação):



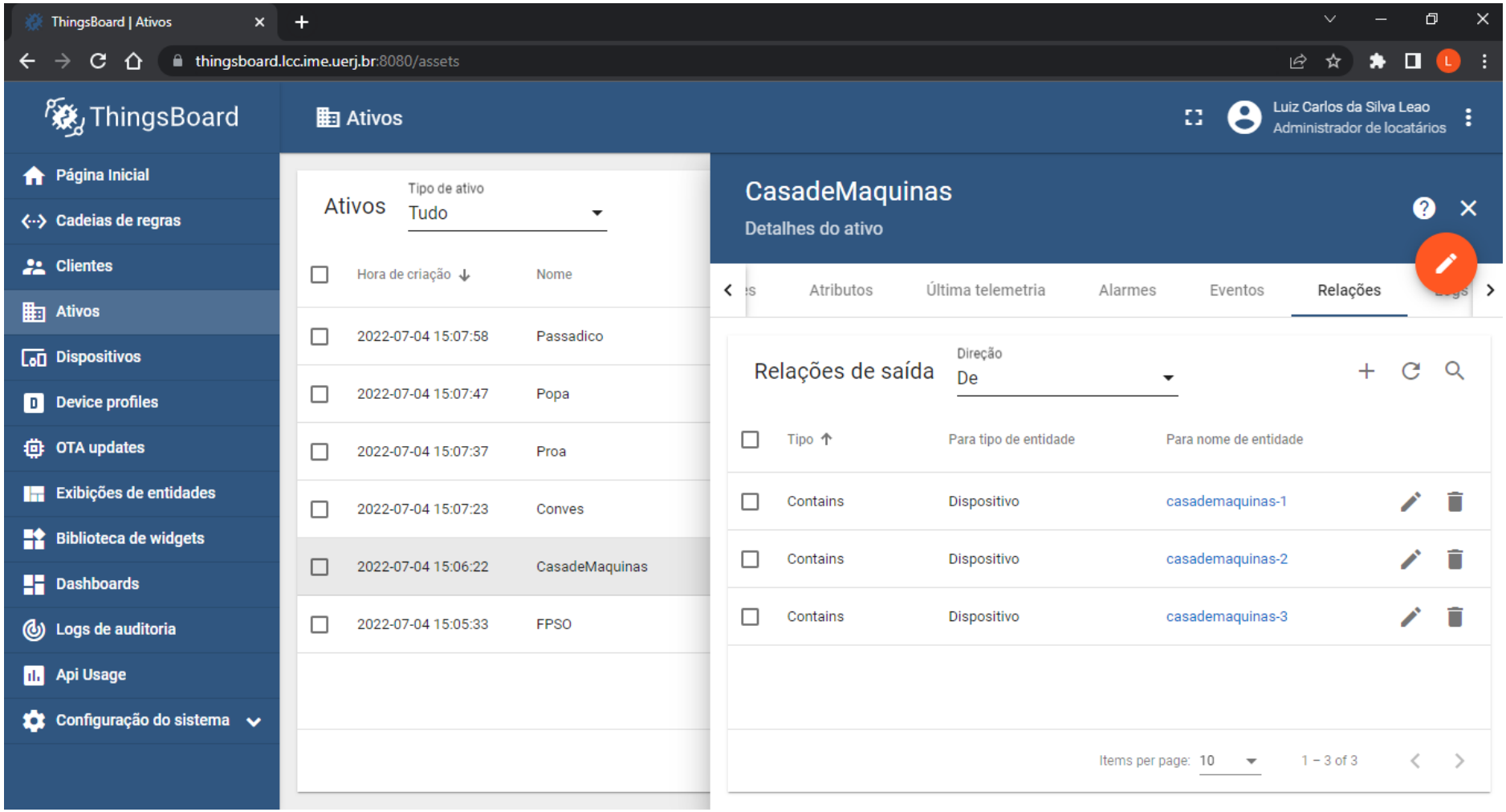
m) Dispositivos (um total de 14 simulados por um Raspberry Pi e 1 NodeMCU + DHT 22) – medidas: temperatura e humidade



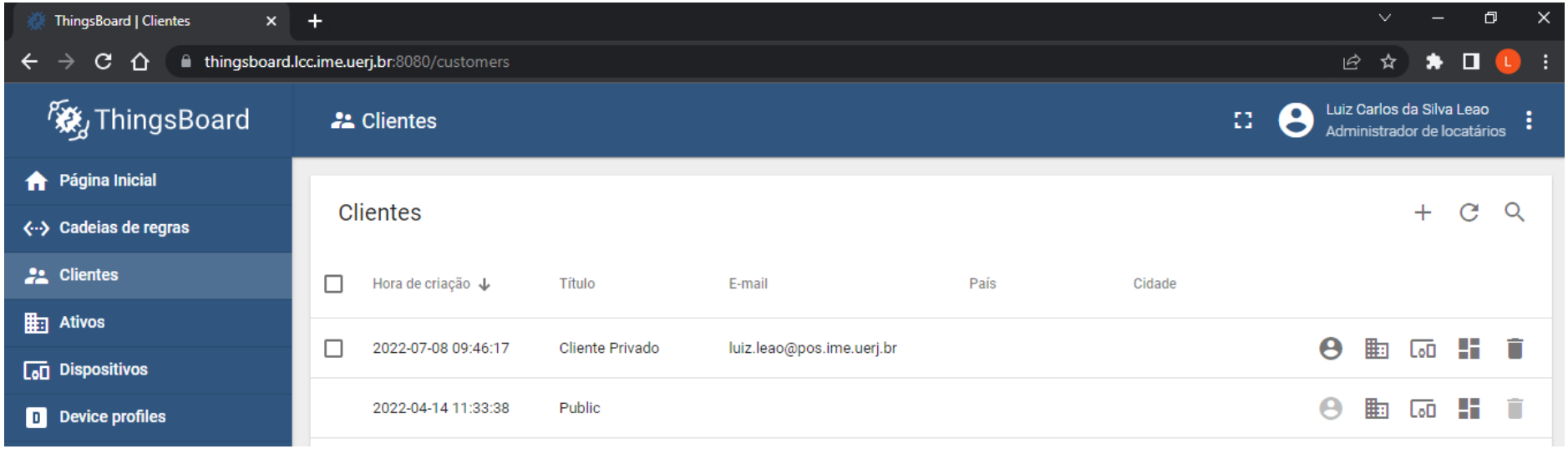
n) Assets: 1 agregando os sensores de cada área da plataforma

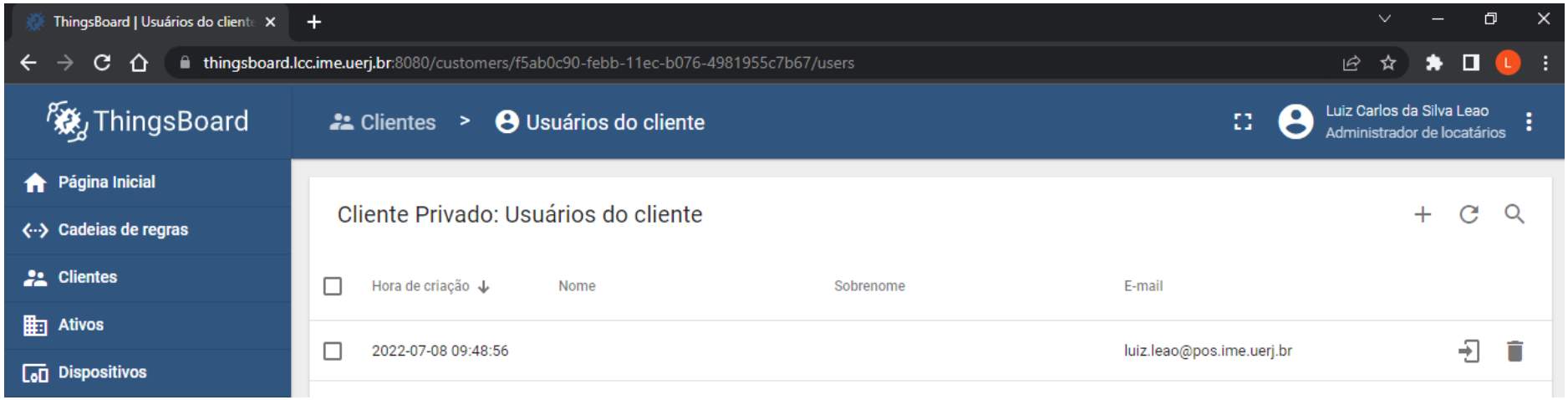


o) Relações dos Assets:

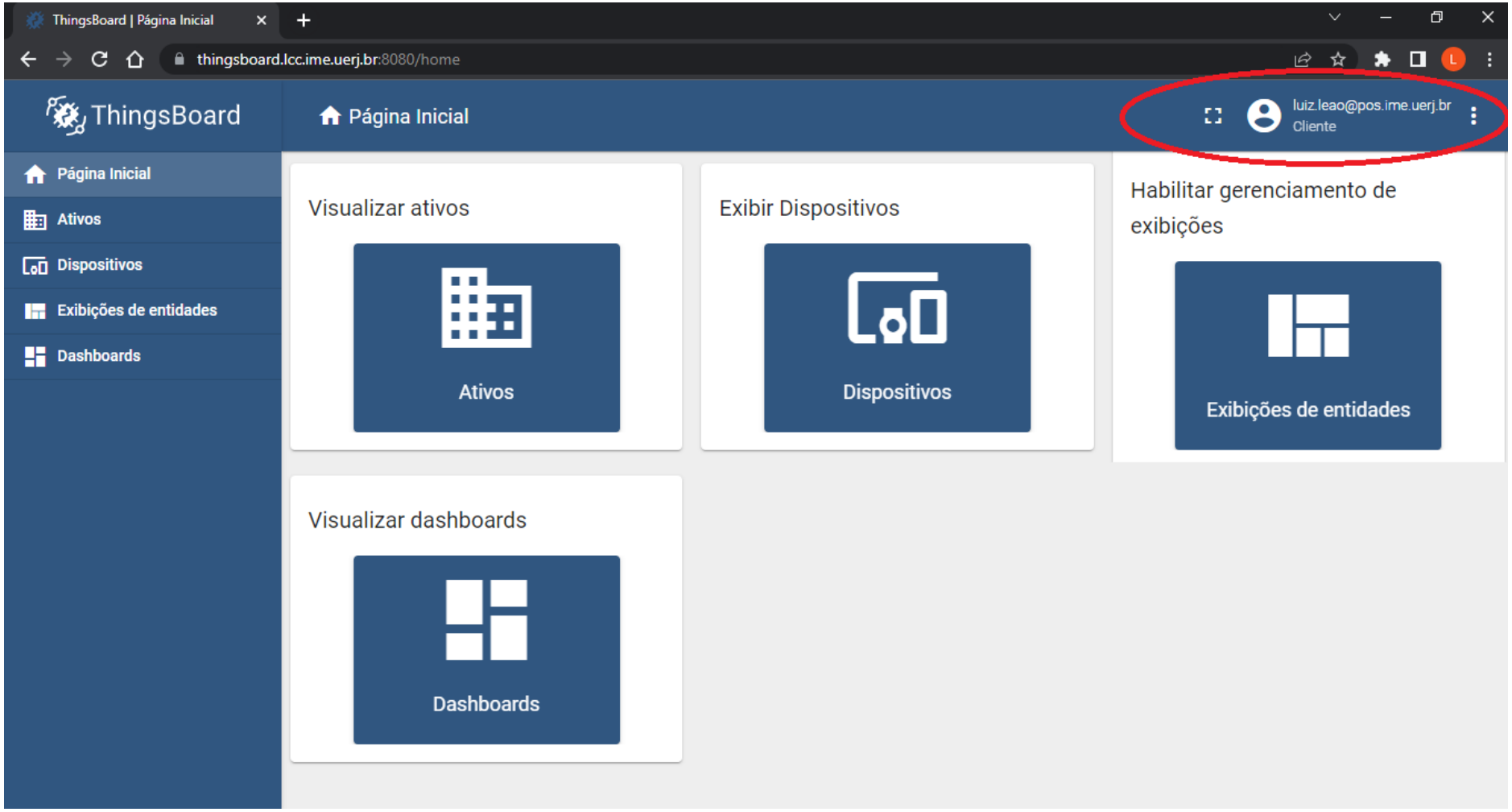


p) Clientes:

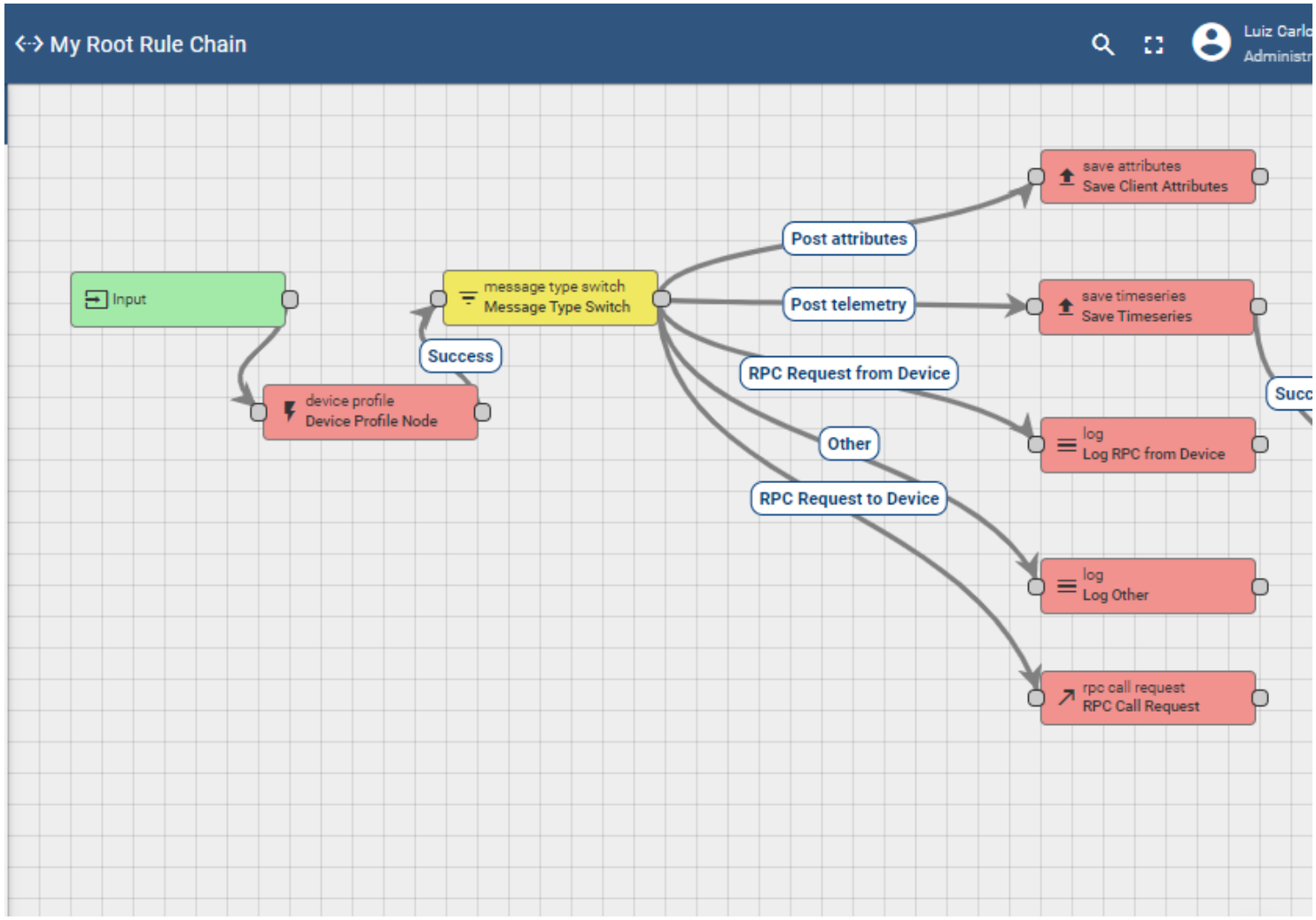

ThingsBoard | Clientes
thingsboard.lcc.ime.uerj.br:8080/customers
ThingsBoard
Clientes
Luiz Carlos da Silva Leao
Administrador de locatários
Página Inicial
Cadeias de regras
Clientes
Ativos
Dispositivos
Device profiles
Clientes
Hora de criação
Título
E-mail
País
Cidade
2022-07-08 09:46:17
Cliente Privado
luiz.leao@pos.ime.uerj.br
2022-04-14 11:33:38
Public


ThingsBoard | Usuários do cliente
thingsboard.lcc.ime.uerj.br:8080/customers/f5ab0c90-febb-11ec-b076-4981955c7b67/users
ThingsBoard
Clientes > Usuários do cliente
Luiz Carlos da Silva Leao
Administrador de locatários
Página Inicial
Cadeias de regras
Clientes
Ativos
Dispositivos
Cliente Privado: Usuários do cliente
Hora de criação
Nome
Sobrenome
E-mail
2022-07-08 09:48:56
luiz.leao@pos.ime.uerj.br

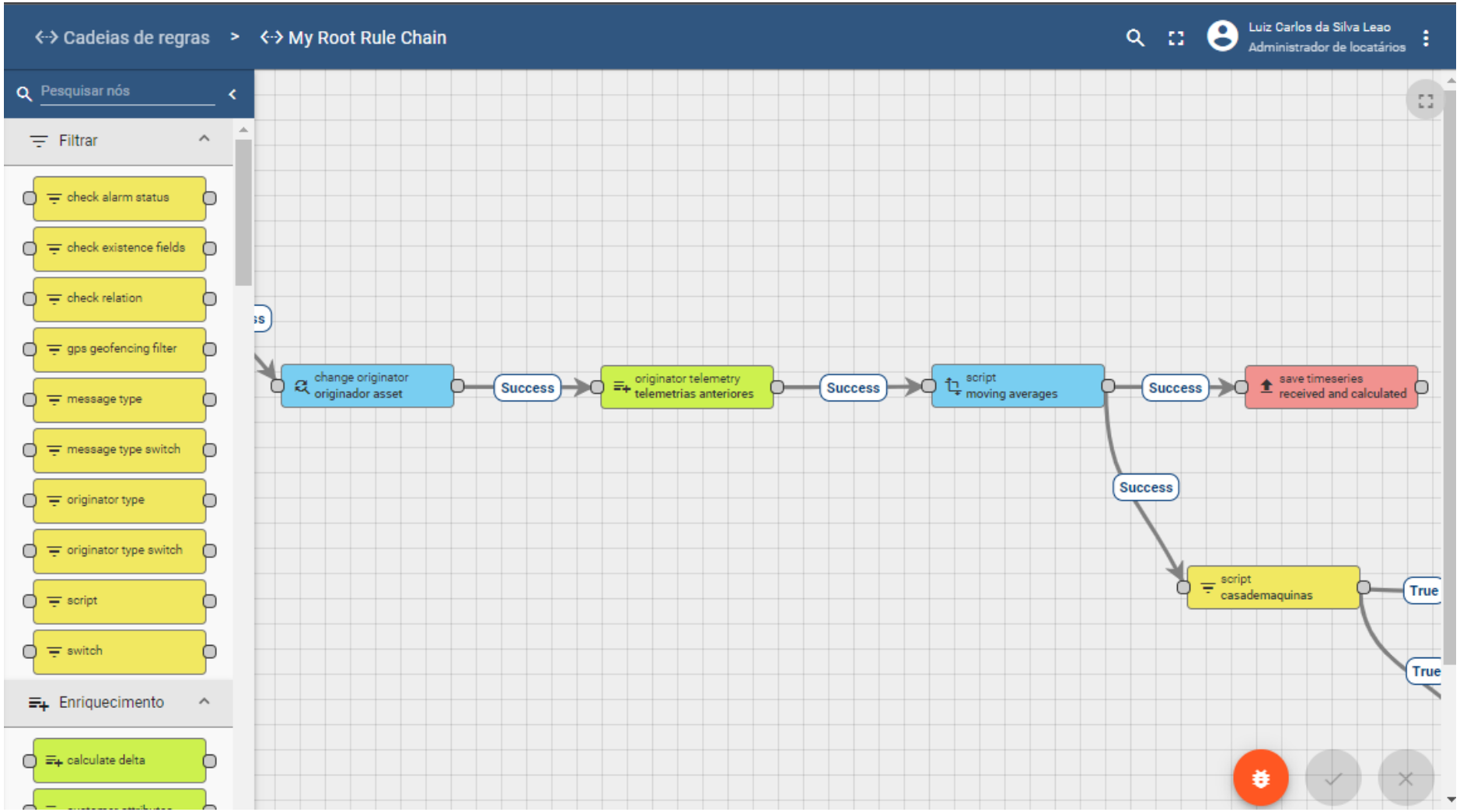
q) Criação de cliente privado que só tem perfil de leitura do dashboard:



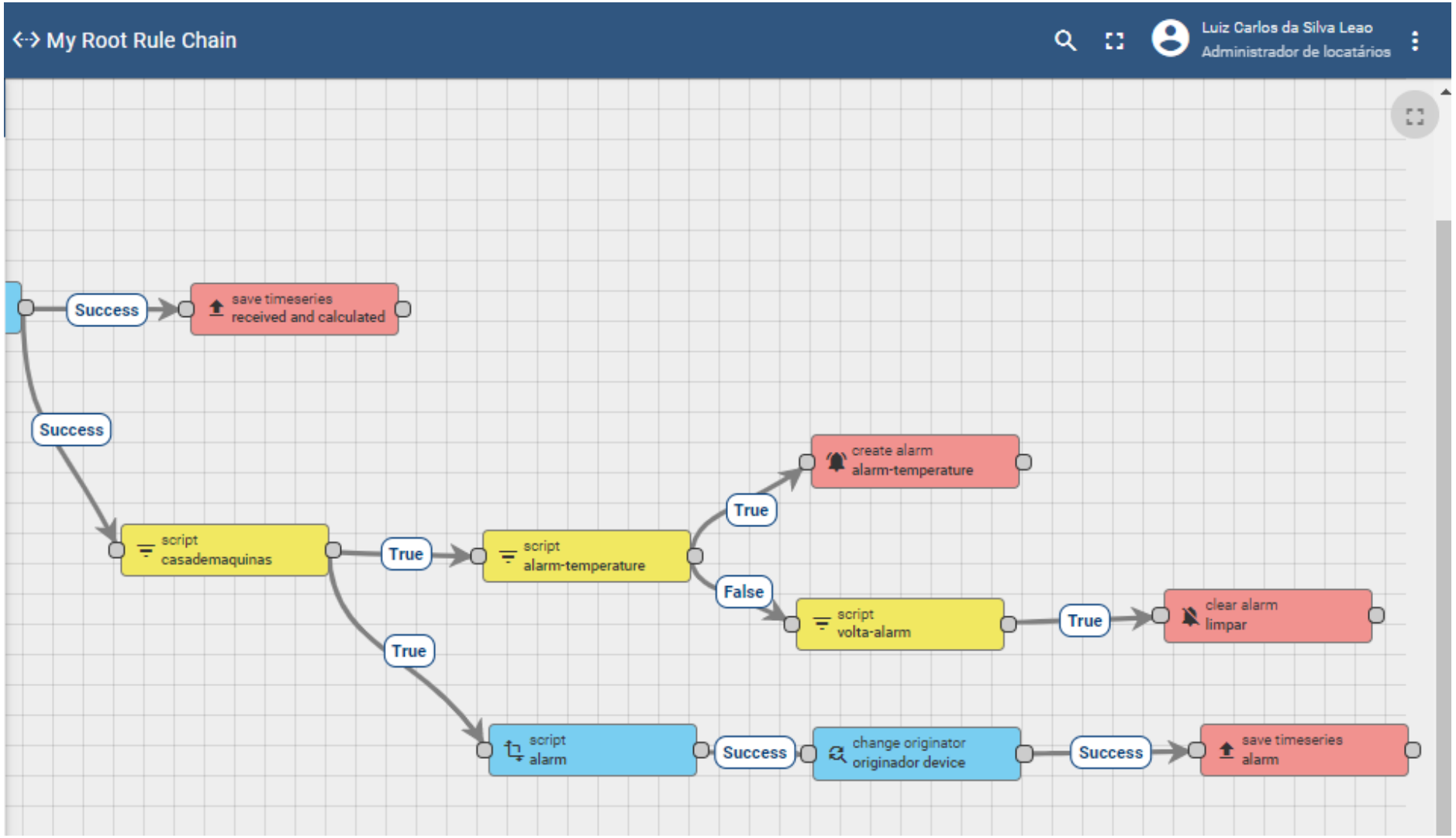
r) Cadeia de Regras (My Root Rule Chains) – uma cadeia de regras única para todo o projeto:



s) Cadeia de Regras (Continuação):



t) Cadeia de Regras (final):



u) Scripts (1 – Média e Média Móvel):

```
//return {msg: msg, metadata: metadata, msgType: msgType};
var newMsg = {};
var tname = "";
var tempArray = [];
var newMsg = {}
var media = 0.0;
var average = 0.0;
var i =0;
media = parseFloat((msg.temperature).toFixed(2));
tname = "Temperature" +
metadata.deviceName.slice(metadata.deviceName.indexOf("-"));
newMsg[tname] = media;
tempArray = [];
average = media;
if(metadata.hasOwnProperty(tname)){
   tempArray = JSON.parse(metadata[tname]);
   //calcula a média das 5 primeiras amostras de cada sensor e da
média dos sensores
   for(var j = 0;j<tempArray.length && j<4;j++){
      average = (average*(j+1) +
parseFloat(tempArray[j].value))/(j+2);
   }
}
newMsg["media" + tname] =
parseFloat(average.toFixed(2));//adiciona a média móvel do
sensor como nova telemetria
for(var key in metadata){
   if(key.startsWith("Temperature")){
      if(key !== tname){
         tempArray = [];
         tempArray = JSON.parse(metadata[key]);
         media = (media*(i+1) +
parseFloat(tempArray[0].value))/(i+2);
         i++;
      }
   }
}
media = parseFloat(media.toFixed(2));
newMsg.media = media;
tempArray = [];
average = media;
if(metadata.hasOwnProperty("media")){
   tempArray = JSON.parse(metadata.media);
   //calcula a média das 5 primeiras amostras de cada sensor e da
média dos sensores
   for(var j = 0;j<tempArray.length && j<4;j++){
      average = (average*(j+1) +
parseFloat(tempArray[j].value))/(j+2);
   }
}
newMsg.mediaMovel = parseFloat(average.toFixed(2));//adiciona
a média móvel do sensor como nova telemetria
var newMeta = {};
```

```
for(var key in metadata) {
  //mantém os metadados do sensor e o timestamp da
mensagem
  if(key.startsWith("device") || key.startsWith("ts")){
    newMeta[key] = metadata[key];
  }
}
return {msg: newMsg, metadata: newMeta, msgType: msgType};
```

v) Scripts (Alarme):

```
return metadata.deviceName.startsWith("casademaquinas");

var newMsg = {"alarm" : (msg.mediaMovel > 32)};
metadata.ts = new Date().getTime();
return {msg: newMsg, metadata: metadata, msgType: msgType};

return msg.mediaMovel > 32;
```

w) Script (arquivo do Arduino *.ino para envio da telemetria do NodeMCU + DHT22 e conexão RPC para liga/desliga do LED)

https-NodeMCU-DHT22.ino:

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecureBearSSL.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

ESP8266WiFiMulti WiFiMulti;

const char* REDE_WIFI = "***";
const char* SENHA_WIFI = "***";
```

```
//UERJ
String SERVER_URL =
"https://thingsboard.lcc.ime.uerj.br:8080/api/v1/";
String TOKEN = "***";

// fingerprint do servidor Thingsboard da UERJ para conexão
HTTPS
const uint8_t FINGERPRINT[20] =
{0xd8,0xae,0xeb,0x77,0x30,0xe3,0xa8,0x2a,0x38,0x4b,0x56,0xd8,
0x6d,0xa8,0xf9,0x0a,0x39,0x41,0x5d,0x0f};

float temp = 0.0;
float umid = 0.0;

int porta = D1; //porta do LED

unsigned long timeSinceLastRead = 0;//timestamp da última
leitura dos sensores
unsigned long timeSinceLastSend = 0;//timestamp do último
envio das telemetrias
unsigned long timeSinceLastRpc = 0;//timestamp da última
verificação dos comandos RPC
const long intervalSend = 30000;//envio a cada 30 segundos

void setup() {
  Serial.begin(115200);
  Serial.setTimeout(2000);

  while(!Serial) { }
  dht.begin();

  // espera 4s para a conexão
  for (uint8_t t = 4; t > 0; t--) {
    Serial.printf("[SETUP] Aguarde %d...\n", t);
    Serial.flush();
    delay(1000);
  }

  //conexão wifi
  WiFi.mode(WIFI_STA);
  //WiFiMulti.addAP("RapNet_Virtua", "a0abea1234");
  WiFiMulti.addAP(REDE_WIFI, SENHA_WIFI);
  //client.setCallback(on_message);
  pinMode(porta,OUTPUT);
  digitalWrite(porta,HIGH);//começa com o LED ligado
  while(WiFiMulti.run() != WL_CONNECTED){}
  Serial.println("Dispositivo iniciado e WIFI conectada");
}

void loop() {
  unsigned long timeCurrent = millis();
  //Somente captura os valores dos sensores
  if((timeCurrent - timeSinceLastRead) >= 2000) {
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    float f = dht.readTemperature(true);
```

```
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    timeSinceLastRead = timeCurrent - 1000;
    return;
  }
  temp=t;
  umid=h;
  /*
  if(temp<20){
    temp=20;
  }else if(temp>=5){
    temp=20;
  }else{
    temp+=1;
  }
  */
  timeSinceLastRead = timeCurrent;
  Serial.println("Telemetrias atualizadas");
 }

 // wait for WiFi connection
 if ((WiFiMulti.run() == WL_CONNECTED)) {
  std::unique_ptr<BearSSL::WiFiClientSecure>client(new
BearSSL::WiFiClientSecure);

  client->setFingerprint(FINGERPRINT);
  //client->setInsecure();

  HTTPClient https;
```

```
  String payload = "";
  String url_conexao;
  int httpCode;
  if((timeCurrent - timeSinceLastRpc) >= 5000){
    //Verificação dos comandos RPC
    //concatena o token na url de conexão
    url_conexao = SERVER_URL + TOKEN + "/rpc?timeout=5000";
    //url_conexao.replace("$TOKEN$",TOKEN);
    Serial.print("url_conexao:" + url_conexao + "\n");

    //if (https.begin(*client,
"https://thingsboard.lcc.ime.uerj.br:8080/api/v1/sHxl9MKNW8O
BqneP3caV/telemetry")) {  // HTTPS
    if (https.begin(*client, url_conexao)) {  // HTTPS

      Serial.print("[HTTPS] GET Payload: " + payload + "\n");

      httpCode = https.GET();

      if (httpCode > 0) {

        Serial.printf("[HTTPS] GET... code: %d\n", httpCode);

        if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY) {
          payload = https.getString();
          https.end();
          Serial.println(payload);
```

```
        DynamicJsonDocument
json(uint32_t(float(payload.length() + 10)/10)*10);
        char payloadchar[payload.length()+1];
        strncpy(payloadchar,payload.c_str(),payload.length());
        DeserializationError error =
deserializeJson(json,payloadchar);
        if(error){
         Serial.println("Payload não está no formato JSON!");
        }else{
         if(json["method"].isNull()){
          Serial.println("Método não encontrado");
         }else{
          if(json["method"].as<String>()=="getValor"){
           if(!json["id"].isNull()){
            int id = json["id"].as<int>();
            //Retorna
            url_conexao = SERVER_URL + TOKEN + "/rpc/" +
String(id);
            if (https.begin(*client, url_conexao)) {  // HTTPS

             httpCode = https.POST(digitalRead(porta) ? "true" :
"false");

             if (httpCode > 0) {

              Serial.printf("[HTTPS] POST... code: %d\n",
httpCode);
              if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY) {
               payload = https.getString();
               Serial.println(payload);
              }
             } else {
              Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
             }
             https.end();
            }
           }
          }else if(json["method"].as<String>()=="setValor"){
           if(json["params"].isNull()){
            Serial.println("Parâmetros não encontrados");
           }else{

digitalWrite(porta,json["params"].as<String>()=="true"?HIGH:LO
W);
            timeSinceLastSend = 0;
           }
          }
         }
        }else{
         Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
         https.end();
        }
```

```
      } else {
        Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
      }
    } else {
      Serial.printf("[HTTPS] Unable to connect\n");
    }
    timeSinceLastRpc = timeCurrent;
  }

  if((timeCurrent - timeSinceLastSend) >= intervalSend){
    //Envio das telemetrias a cada intervalSend
    //concatena o token na url de conexão
    url_conexao = SERVER_URL + TOKEN + "/telemetry";
    //url_conexao.replace("$TOKEN$",TOKEN);
    Serial.print("url_conexao:" + url_conexao + "\n");

    //if (https.begin(*client,
"https://thingsboard.lcc.ime.uerj.br:8080/api/v1/sHxl9MKNW8O
BqneP3caV/telemetry")) {  // HTTPS
    if (https.begin(*client, url_conexao)) {  // HTTPS

      Serial.print("[HTTPS] POST...\n");

      payload = "{\"temperature\":"+ String(temp) +",
\"humidity\":" + String(umid) + ",\"port\":" + (digitalRead(porta) ?
"true" : "false") + "}";

      Serial.print("[HTTPS] Payload: " + payload + "\n");

      httpCode = https.POST(payload);

      if (httpCode > 0) {

        Serial.printf("[HTTPS] POST... code: %d\n", httpCode);

        if (httpCode == HTTP_CODE_OK || httpCode ==
HTTP_CODE_MOVED_PERMANENTLY) {
          payload = https.getString();
          Serial.println(payload);
        }
      } else {
        Serial.printf("[HTTPS] GET... failed, error: %s\n",
https.errorToString(httpCode).c_str());
      }

      https.end();

    } else {
      Serial.printf("[HTTPS] Unable to connect\n");
    }
    timeSinceLastSend = timeCurrent;
  }
 }
}
```

x) Script Python para rodar no Raspberry Pi e simular o envio de telemetria para o ThingsBoard (replicou-se o mesmo para os demais sensores):

```
import os
import time
import sys
import paho.mqtt.client as mqtt
import json
from random import randint
import random
THINGSBOARD_HOST = 'thingsboard.lcc.ime.uerj.br'
ACCESS_TOKEN = 'xyz'

# configure aqui o intervalo
INTERVAL=36

sensor_data = {'temperature': 0, 'humidity': 0}

next_reading = time.time()

client = mqtt.Client()

# Set access token
client.username_pw_set(ACCESS_TOKEN)

# Connect to ThingsBoard using default MQTT port and 60
seconds keepalive interval
client.connect(THINGSBOARD_HOST, 1883, 60)

client.loop_start()

try:
   while True:
      # gera um valor de umidade aleatório
      humidity = randint(70,80)
      humidity = round(humidity, 2)
      # gera um valor de temperatura aleatório
      temperature = randint(25,40)
      temperature = round(temperature, 2)

      print(u"Temperatura: {:g}\u00b0C, Umidade:
{:g}%".format(temperature, humidity))
      sensor_data['temperature'] = temperature
      sensor_data['humidity'] = humidity

      # Sending humidity and temperature data to ThingsBoard
      client.publish('v1/devices/me/telemetry',
json.dumps(sensor_data), 1)

      next_reading += INTERVAL
      sleep_time = next_reading-time.time()
      if sleep_time > 0:
         time.sleep(sleep_time)
```

```
except KeyboardInterrupt:
    pass
```

```
client.loop_stop()
client.disconnect()
```