Problema 3
IoT: Internet das Coisas

Universidade Estadual de Feira de Santana
Curso: Engenharia de Computação
TEC 499 - MI - Sistemas Digitais
Professor: Thiago Cerqueira de Jesus

Equipe

- Daniel Lucas Alves Ferreira de Jesus
- Ian Zaque Pereira de Jesus dos Santos

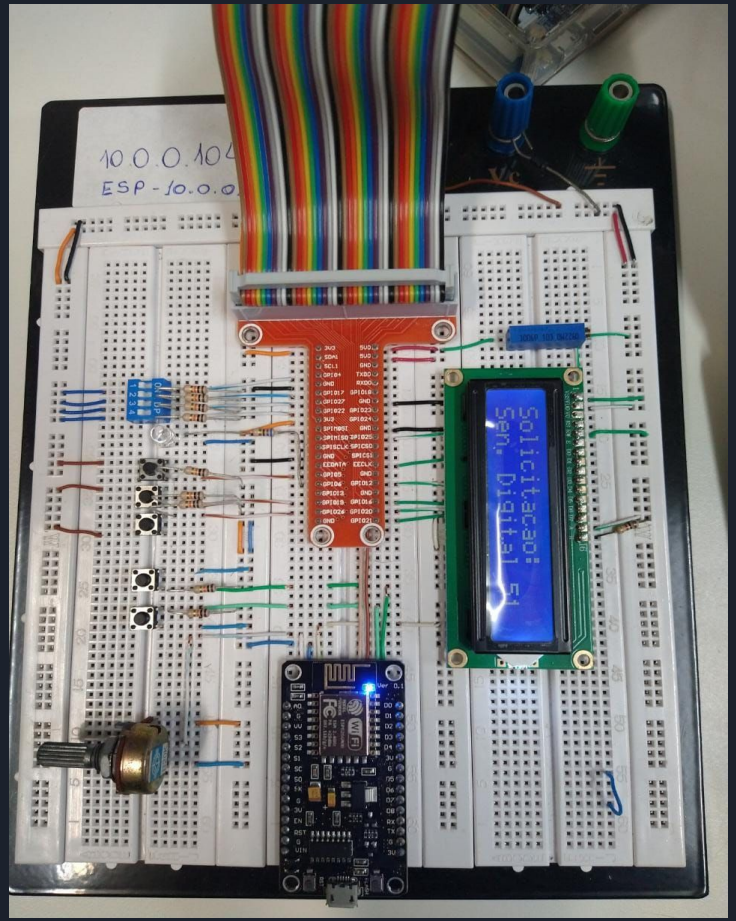


Imagem 1. Orange Pi, Display e NodeMCU.

Hardware - SBC

- Orange Pi PC Plus
- H3 Quad-core Cortex-A7 H.265/HEVC 4K
- Mali400MP2 GPU @600MHz
- SDRAM: 1GB DDR3
- 8GB eMMC Flash
- Alimentação via USB OTG
- Header de 40 pinos
- LED de alimentação, LED de status
- Suporte a Android, Lubuntu e Debian

Hardware - Orange Pi

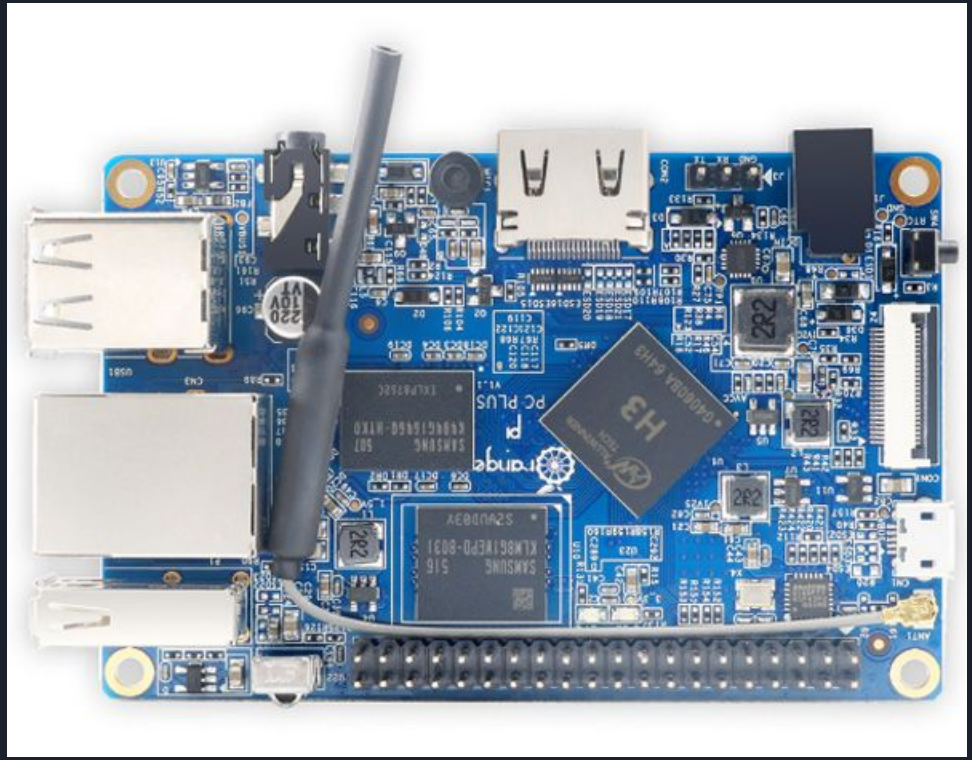
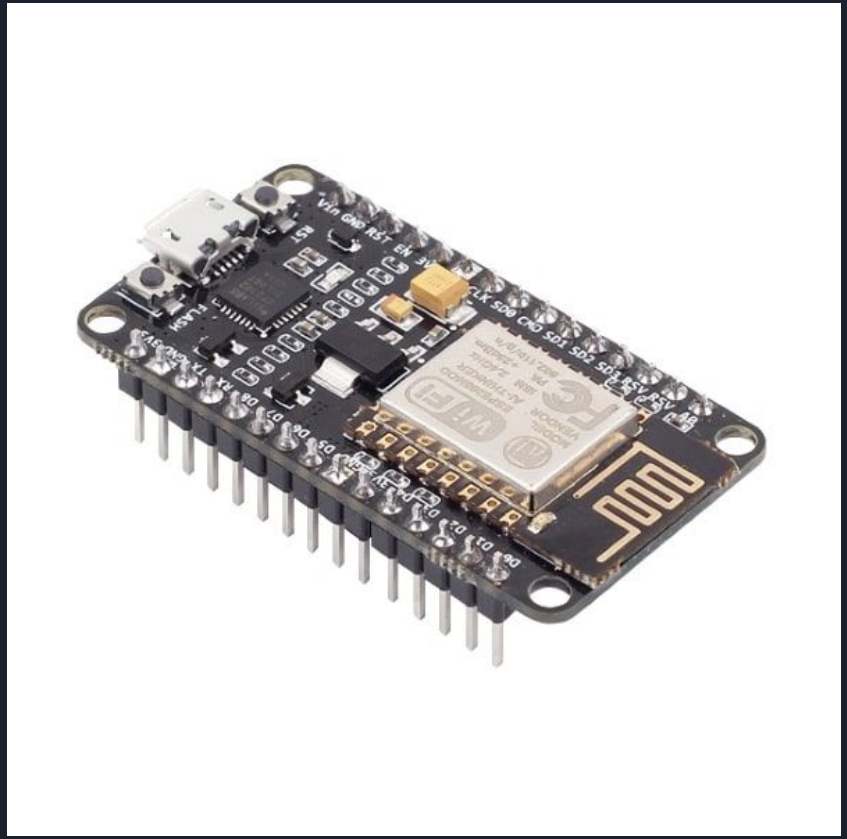


Imagem 2. Orange Pi PC Plus.

Hardware - NodeMCU

- ESP8266 NodeMcu ESP-12E Module
- Wireless padrão 802.11 b/g/n
- Antena embutida
- Conector micro-usb
- Modos de operação: STA/AP/STA+AP
- Portas GPIO: 11
- GPIO com funções de PWM, I2C, SPI, etc
- Tensão de operação: 4,5 ~ 9V
- Taxa de transferência: 110 a 460800 bps
- Dimensões: 49 x 25,5 x 7 mm



Etapas do algoritmo - SBC

1. Definição de bibliotecas, variáveis globais, estruturas e protótipos de funções.
2. Definição de variáveis locais, configuração MQTT e LCD Display.
3. Inicialização IHM Display, IHM Terminal.
4. Envio e recebimento automático de comandos e respostas via MQTT.
5. Verificação e tratamento de respostas.
6. Atualização de histórico de medições dos sensores.
7. Publicação de dados dos sensores e NodeMCU para IHM Web.

Etapas do algoritmo - NodeMCU

1. Definição de bibliotecas, variáveis globais e funções.
2. Configuração WiFi, MQTT e GPIO na função setup();.
3. Inicialização WiFi e MQTT.
4. Aguarda recebimento de comandos da SBC na função loop();.
5. Envio de respostas MQTT, mediante comandos.

Etapas do algoritmo - IHM Web - API

1. Configuração dos Servidores - MQTT e HTTP
2. Configuração dos tópicos MQTT
3. Armazena as mensagens nas bases de dados específicas
4. Definição das rotas do servidor HTTP
5. Acesso a bases de dados via rotas definidas

Etapas do algoritmo - IHM Web - Interface HTML

1. Biblioteca de acesso ao servidor HTTP (axios)
2. Requisição de dados via click dos Botões
    a. Realiza requisições a cada X segundos
3. Recebe os dados e configura os mesmos
4. Monta os gráficos com os dados

SBC - Definição de bibliotecas e variáveis globais

```
#include <lcd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/time.h>

//BIB FOR MQTT
#include <MQTTClient.h>

//BIB FOR INTERRUPT, GPIO
#include <wiringPi.h>

#define MQTT_ADDRESS    "tcp://10.0.0.101:1883"
#define USER            "aluno"
#define PASSWORD        "@luno*123"
#define CLIENTID        "MQTTClientRASP"
#define QOS             2
#define TIMEOUT         1000L

#define NODEMCU_PUBLISH   "NODEMCU_PUBLISH"
#define NODEMCU_RECEIVE   "NODEMCU_RECEIVE"
#define SENSORS_HISTORY   "SENSORS_HISTORY"
#define STATUS_NODEMCU    "STATUS_NODEMCU"

// Pinos do Display LCD
#define RS  13
#define E   18
#define D4  21
#define D5  24
#define D6  26
#define D7  27
```

Imagem 3. Bibliotecas e variáveis globais.

SBC - Definição de variáveis globais e funções

```
MQTTClient client;

int display_lcd;

You, 2 days ago | 1 author (You)
typedef struct Historic {
  char *sensor;
  int historic[10];
  int timestamps[10];
  int last_modified;
} Historic;

// ONE HISTORY FOR EACH SENSOR
Historic array_registros[10];

void publish(MQTTClient client, char* topic, char* payload);
int on_message(void *context, char *topicName, int topicLen, MQTTClient_message *message);
void reconnect(void *context, char *cause);
void evaluateRecData(char *topicName, char *payload);
char *substring(char *src, int start, int end);
void updateHistory(char * sensor, int newValue);
char *createJson(int index);
void initArrayRegistros();
void initDisplay();
void write_textLCD(char *linha1, char *linha2);
```

Imagem 4. Variáveis globais e funções.

SBC - Função main

```c
int main(int argc, char *argv[]){
    int rc;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_create(&client, MQTT_ADDRESS, CLIENTID, MQTTCLIENT_PERSISTENCE_NONE, NULL);
    MQTTClient_setCallbacks(client, &conn_opts, reconnect, on_message, NULL);

    conn_opts.username = USER;
    conn_opts.password = PASSWORD;
    rc = MQTTClient_connect(client, &conn_opts);

    if (rc != MQTTCLIENT_SUCCESS){
        printf("\n\rFalha na conexao ao broker MQTT. Erro: %d\n", rc);
        exit(-1);
    }

    MQTTClient_subscribe(client, NODEMCU_RECEIVE, 0);           //Subscribing to MQTT topic that NodeMcu publishes

    wiringPiSetup();
    initDisplay();              // INIT LCD DISPLAY
    initArrayRegistros();      // INIT HISTORIC
```

Imagem 5. Função principal, configuração e inicialização MQTT, Display e históricos.

SBC - Inicialização IHM Terminal

```
system("cls || clear");
write_textLCD("  PBL 3 - SD  ", "      IoT      ");

printf("PBL 3 - IoT: A Internet das Coisas. \n \n");
printf("As a��es seguir�o a seguinte ordem: \n");
printf("1 - Solicita a situa��o atual do NodeMCU. \n");
printf("2 - Solicita o valor da entrada anal�gica. \n");
printf("3 - Solicita o valor de uma das entradas digitais. \n");
printf("4 - Acendimento do led da NodeMCU. \n");
printf("5 - Desligamento do led da NodeMCU. \n \n");
sleep(2);

for(int i = 0; i <= 4; i++){           // LOOP TO ENGAGE AUTOMATIC ACTIONS
        if(i == 0){                   //this checks the NodeMCU status.
            printf("A��o: Solicita a situa��o atual do NodeMCU. \n");
            write_textLCD("Solicitacao:", "Status NodeMCU");
            sleep(5);

            publish(client, NODEMCU_PUBLISH, "30");
        }

        if(i == 1){                   ///this requests analog input value
            system("cls || clear");
            printf("A��o: Solicita o valor da entrada anal�gica. \n");
            write_textLCD("Solicitacao:", "Sen. Analogico");
            sleep(5);

            publish(client, NODEMCU_PUBLISH, "40");
        }
```

Imagem 6. Envio automático de comandos para tópico de publicação.

SBC - Comandos de sensores

```
if(i == 1){                    ///this requests analog input value
    system("cls || clear");
    printf("A��o: Solicita o valor da entrada anal�gica. \n");
    write_textLCD("Solicitacao:", "Sen. Analogico");
    sleep(5);

    publish(client, NODEMCU_PUBLISH, "40");
}

if(i == 2){                    //this requests some digital input value
    system("cls || clear");
    printf("A��o: Solicita o valor de uma das entradas digitais. \n");

    //LOOP TO COMMUTE DIGITAL SENSORS REQUESTS
    for (int idxSensor = 50; idxSensor < 59; idxSensor++){
        sleep(1);
        printf("Sensor Digital: %d. \n \n", idxSensor);

        char numStr[6] = "";
        sprintf(numStr, "%d", idxSensor);              //CONVERT INTEGER TO STRING

        char text[15];
        strcpy(text, "Sen. Digital ");
        strcat(text, numStr);
        write_textLCD("Solicitacao:", text);

        sleep(5);
        publish(client, NODEMCU_PUBLISH, numStr);
    }
}
```

Imagem 7. Solicitação de respostas dos sensores da NodeMCU.

SBC - Comandos de LED

```
if(i == 3){                    //this turn on the led
    system("cls || clear");
    printf("A??o: Acendimento do LED da NodeMCU. \n");
    write_textLCD("Solicitacao:", "Ligar LED");
    sleep(5);

    publish(client, NODEMCU_PUBLISH, "60");
}

if(i == 4){                    //this turn off the led
    i = -1;              // RESET THE LOOP

    system("cls || clear");
    printf("A??o: Desligamento do LED do NodeMCU. \n");
    write_textLCD("Solicitacao:", "Desligar LED");
    sleep(5);

    publish(client, NODEMCU_PUBLISH, "70");
    system("cls || clear");
}
```

Imagem 8. Solicitação de interação com LED da NodeMCU.

SBC - Funções MQTT

```c
// METHOD THAT WRITES IN THE TOPIC SOME MESSAGE
// Param: CLIENT (instance of MQTTClient), TOPIC (where to write), PAYLOAD (message to be written)
// Return: void
void publish(MQTTClient client, char* topic, char* payload) {
    MQTTClient_message pubmsg = MQTTClient_message_initializer;

    pubmsg.payload = payload;
    pubmsg.payloadlen = strlen(pubmsg.payload);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;
    MQTTClient_deliveryToken token;
    MQTTClient_publishMessage(client, topic, &pubmsg, &token);
    MQTTClient_waitForCompletion(client, token, TIMEOUT);
}

// METHOD CALLED WHENEVER A MESSAGE IS RECEIVED IN ANY SUBSCRIBED TOPIC
// Return: 1
int on_message(void *context, char *topicName, int topicLen, MQTTClient_message *message) {
    char* payload = message->payload;
    evaluateRecData(topicName, payload);

    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return 1;
}

// METHOD TO AUTO RECONNECT IF MQTT IS DOWN
// PARAM: CONTEXT (data from MQTT_OPTS), CAUSE (why caiu)
void reconnect(void *context, char *cause){
    printf("Broker foi desconectado... Tentando novamente \n \n");
    int rc = MQTTClient_connect(client, context);

    if (rc != MQTTCLIENT_SUCCESS){
        printf("\n\rFalha na conexao ao broker MQTT. Erro: %d\n", rc);
        exit(-1);
    }
    else { printf("Reconectado\n"); }

    MQTTClient_subscribe(client, NODEMCU_RECEIVE, 0);
}
```

Imagem 9. Funções MQTT de publicação, recebimento e reconexão do broker.

SBC - Funções MQTT

```
void evaluateRecData(char * topicName, char *payload){

    if(strcmp(topicName, "NODEMCU_RECEIVE") == 0){
        if (payload[0] == '1' && payload[1] == 'F'){                    // 1F
            write_textLCD("Resposta: ", "NodeMCU not ok");
            publish(client, STATUS_NODEMCU, "NodeMCU not Ok");
         }

        else if (payload[0] == '0' && payload[1] == '0'){               // 00
            write_textLCD("Resposta: ", "NodeMCU Ok");
            publish(client, STATUS_NODEMCU, "NodeMCU Ok");
        }

        else if (payload[0] == '0' && payload[1] == '1'){               // 01
            char *analogInputValue = substring(payload, 2, sizeof(payload)+1);     // COPYING

            int value = atoi(analogInputValue);             // CONVERT STRING TO INT
            updateHistory("A0", value);
            write_textLCD("Sen. Analogico:", analogInputValue);
        }

        else if (payload[0] == '0' && payload[1] == '2'){               //02
            // COPYING THE DIGITAL SENSOR & INPUT VALUE RECEIVED. Ex.: 02A01234
            char *digitalSensor = substring(payload, 2, 4);             // A0
            char *digitalInputValue = substring(payload, 4, sizeof(payload)*sizeof(payload));

            int value = atoi(digitalInputValue);          // CONVERT STRING TO INT
            char text[15];
            strcpy(text, "Sen. Digital ");
            strcat(text, digitalSensor);                 // Something like: Sen. Digital D5

            write_textLCD(text, digitalInputValue);
            updateHistory(digitalSensor, value);
        }

        else{ write_textLCD("Resposta:", "NodeMCU not ok"); }
    }
    return;
```

Imagem 10. Função de verificação de respostas MQTT.

SBC - Função Substring

```c
char *substring(char *src, int start, int end){
    int len = end - start;
    char *dest = (char*)malloc(sizeof(char) * (len + 1));    /
    strncpy(dest, (src + start), len);                       /
    return dest;
}
```

Imagem 11. Função de criação de substring a partir de index start até index end.

SBC - Função UpdateHistory

```
void updateHistory(char *sensor, int newValue){
    int idx = 0;
    struct timeval now;                         // timestamp
    gettimeofday(&now, NULL);

    char *json = (char*)malloc( (sizeof(char) * (200)) + 1 );
    char sensorTopic[20];
    strcpy(sensorTopic, "SENSORS_HISTORY/");
    strcat(sensorTopic, sensor);

    if(strcmp(sensor, "A0") == 0){
       idx = ++array_registros[0].last_modified;
       if(array_registros[0].last_modified >= 9){ array_registros[0].last_modified = -1; }

       array_registros[0].historic[idx] = newValue;
       array_registros[0].timestamps[idx] = now.tv_sec;
       strcpy(json, createJson(0));
       publish(client, sensorTopic, json);
    }

    else if(strcmp(sensor, "D0") == 0){
       idx = ++array_registros[1].last_modified;
       if(array_registros[1].last_modified >= 9){ array_registros[1].last_modified = -1; }

       array_registros[1].historic[idx] = newValue;
       array_registros[1].timestamps[idx] = now.tv_sec;
       strcpy(json, createJson(1));
       publish(client, sensorTopic, json);
    }
```

Imagem 12. Função de atualização de histórico de medições.

SBC - Função CreateJson

```
char *createJson(int index){
    char *json = (char*)malloc((sizeof(char) * (200)) + 1);
    strcpy(json, "{\"historico\": ");           //Ex.: { "historico\":

    // LOOP TO CONCAT THE START OF JSON
    for(int i = 0; i < 10; i++){
        char valueHistoric[20];
        sprintf(valueHistoric, "%d", array_registros[index].historic[i]);

        if(i == 0){
            strcat(json, "\"[");                                  //Ex.: { "\sens
            strcat(json, valueHistoric); //Ex.: { "\sensor\":\"A0\", \"histo
        }
        else if(i == 9){
            strcat(json, ", ");
            strcat(json, valueHistoric);
            strcat(json, "]");
        }
        else{
            strcat(json, ", ");
            strcat(json, valueHistoric);
        }
    }       You, yesterday • JSON OK

    strcat(json, "\", \"timestamps\": ");          //Ex.: { "\sensor\":\"
```

```
    // LOOP TO CONCAT THE TIMESTAMPS TO JSON
    for(int j = 0; j < 10; j++){
        char *valueTimestamp = (char*)malloc((sizeof(char) * (50)) + 1);
        sprintf(valueTimestamp, "%d", array_registros[index].timestamps[j]);

        if(j == 0){
            strcat(json, "\"[");                                  //Ex.: { "\sensor\":\
            strcat(json, valueTimestamp);                         //Ex.: { "\sensor\":\
        }
        else if(j == 9){
            strcat(json, ", ");
            strcat(json, valueTimestamp);
            strcat(json, "]\"}");
        }
        else{
            strcat(json, ", ");
            strcat(json, valueTimestamp);
        }
    }
    return json;
```

Imagens 13 e 14. Função de criação de JSON de histórico de medições.

SBC - Função InitArrayRegistros

```
// METHOD TO INITIALIZE THE VARIABLE 'array_registros'
// Return: void
void initArrayRegistros(){              // [A0, D0, D1, D2, D3, D4, D5, D6, D7, D8]
    array_registros[0].sensor = "A0";
    array_registros[1].sensor = "D0";
    array_registros[2].sensor = "D1";
    array_registros[3].sensor = "D2";
    array_registros[4].sensor = "D3";
    array_registros[5].sensor = "D4";
    array_registros[6].sensor = "D5";
    array_registros[7].sensor = "D6";
    array_registros[8].sensor = "D7";
    array_registros[9].sensor = "D8";

    for(int i = 0; i < 10; i++){
        array_registros[i].last_modified = -1;

        for(int j = 0; j < 10; j++){
            array_registros[i].historic[j] = 0;
            array_registros[i].timestamps[j] = 0;
        }
    }
}
```

Imagem 15. Função de inicialização de histórico de medições.

SBC - Funções Display

```c
/**
 * Realiza as rotinas de inicializacao do display
 */
void initDisplay(){
    display_lcd = lcdInit(2, 16, 4, RS, E, D4, D5, D6, D7, 0, 0, 0, 0) ;
    lcdHome(display_lcd);
    lcdClear(display_lcd);
}

/**
 * Escreve em duas linhas do display LCD
 * @param linha1 - Primeira linha do display
 * @param linha2 - Segunda linha do display
*/
void write_textLCD(char *linha1, char *linha2) {
    lcdHome(display_lcd);
    lcdClear(display_lcd);

    // escreve na primeira linha
    lcdPosition(display_lcd, 0, 0);
    lcdPuts(display_lcd, linha1);

    // escreve na segunda linha
    lcdPosition(display_lcd, 0, 1);
    lcdPuts(display_lcd, linha2);
}
```

Imagem 16. Funções de inicialização e escrita do Display.

NodeMCU - Definições de bibliotecas e variáveis

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include <PubSubClient.h>

#ifndef STASSID
#define STASSID "INTELBRAS"
#define STAPSK  "Pbl-Sistemas-Digitais"
#endif

// topics define
#define TOPICO_SUBSCRIBE "NODEMCU_PUBLISH"
#define TOPICO_PUBLISH "NODEMCU_RECEIVE"
#define ID_MQTT "SDPBL3"
#define TOPICO_SUBSCRIBE2 "nodemcu"
#define BUFFER_SIZE 64
//response command
String operating_normally   = "00";
String with_problem         = "1F";
String analog_entry_measure = "01";
String digital_input_status = "02";

String COMANDO;

//String comand = "";
const char* ssid = STASSID;
const char* password = STAPSK;

// Nome do ESP na rede
const char* host = "ESP-10.0.0.108";
```

```
// Definições de rede
IPAddress local_IP(10, 0, 0, 108);
IPAddress gateway(10, 0, 0, 1);
IPAddress subnet(255, 255, 0, 0);

//settings broker mqtt
const char* mqqt_broker= "10.0.0.101";
const char* mqtt_user= "aluno";
const char* mqtt_password="@luno*123";

//Entradas Digitais
const int sensor_zero = D0;
const int sensor_D1 = D1;
const int sensor_D2 = D2;
const int sensor_D3 = D3;
const int sensor_D4 = D4;
const int sensor_D5 = D5;
const int sensor_D6 = D6;
const int sensor_D7 = D7;
const int sensor_D8 = D8;

// client wifi
WiFiClient espClient;
PubSubClient client(espClient);
```

Imagens 17 e 18. Inclusão e definição de bibliotecas e variáveis globais.

NodeMCU - Função Callback

```
// callback function
void callback(const MQTT::Publish& pub){
 // Serial.print(pub.topic());
   //Serial.println(pub.payload_string());
  //Serial.print(" => ");

  if(pub.has_stream()){
    uint8_t buf[BUFFER_SIZE];
    int read;
    while(read = pub.payload_stream()->read(buf,BUFFER_SIZE)){
      Serial.write(buf,read);
    }
    pub.payload_stream()->stop();
    Serial.println("");
  }else{
       COMANDO = pub.payload_string();  // Recebe o valor DO BROKER
       Serial.println(pub.payload_string());
  }

}
```

Imagens 19. Função Callback para recebimento de dados MQTT.

NodeMCU - Função InitMqtt

```
//initial settings MQTT
void initMqtt(){
  client.set_server(mqqt_broker,1883);
  client.set_callback(callback);

  if(client.connect(MQTT::Connect(ID_MQTT).set_auth(mqtt_user,mqtt_password))){
    Serial.println("Node conectada");
    client.subscribe(TOPICO_SUBSCRIBE);
    client.subscribe(TOPICO_PUBLISH);
  }

}
```

Imagem 20. Função InitMqtt de configuração inicial MQTT.

NodeMCU - Função EnviaEstadoOutputMQTT

```
void EnviaEstadoOutputMQTT(void){
    if(COMANDO == "30"){ client.publish(TOPICO_PUBLISH,operating_normally); }

    else if(COMANDO == "40"){
      String entrada = String(analogRead(A0));
      client.publish(TOPICO_PUBLISH,analog_entry_measure +entrada);
    }

    else if(COMANDO =="50"){
      String entrada_D0 = String(digitalRead(sensor_zero));
      String sensor_D0 = "D0";
      client.publish(TOPICO_PUBLISH,digital_input_status+sensor_D0+entrada_D0);
    }

    else if(COMANDO == "51"){
      String entrada_D1 = String(digitalRead(sensor_D1));
      String sensor_name_D1 = "D1";
      client.publish(TOPICO_PUBLISH,digital_input_status+sensor_name_D1+entrada_D1);
    }

    else if(COMANDO == "52"){
      String sensor_name_D2 = "D2";
      String entrada_D2 = String(digitalRead(sensor_D2));
      client.publish(TOPICO_PUBLISH,digital_input_status+sensor_name_D2+entrada_D2);
    }
```

```
    else if(COMANDO == "57"){
      String sensor_name_D7="D7";
      String entrada_D7 = String(digitalRead(sensor_D7));
      client.publish(TOPICO_PUBLISH,digital_input_status+sensor_name_D7+entrada_D7);
    }

    else if(COMANDO == "58"){
      String sensor_name_D8="D8";
      String entrada_D8= String(digitalRead(sensor_D8));
      client.publish(TOPICO_PUBLISH,digital_input_status+sensor_name_D8+entrada_D8);
    }

    else if(COMANDO=="60"){
      digitalWrite(LED_BUILTIN,LOW);
      client.publish(TOPICO_PUBLISH,operating_normally);
    }

    else if(COMANDO == "70"){
      digitalWrite(LED_BUILTIN,HIGH);
      client.publish(TOPICO_PUBLISH,operating_normally);
    }

    COMANDO="0";
```

Imagens 21 e 22. Função EnviaEstadoOutputMQTT de verificação de comandos MQTT.

NodeMCU - Inicialização WiFi e MQTT

```
// Setup project
pinMode(LED_BUILTIN,OUTPUT);
pinMode(A0,OUTPUT);
pinMode(sensor_zero,OUTPUT);
pinMode(sensor_D1,OUTPUT);
pinMode(sensor_D2,OUTPUT);
pinMode(sensor_D3,OUTPUT);
pinMode(sensor_D4,OUTPUT);
pinMode(sensor_D5,OUTPUT);
pinMode(sensor_D6,OUTPUT);
pinMode(sensor_D7,OUTPUT);
pinMode(sensor_D8,OUTPUT);
Serial.begin(9600);
setup_wifi();
initMqtt();
```

Imagem 23. Inicialização WiFi e MQTT na função loop da NodeMCU.

NodeMCU - Recebimento MQTT

```
void loop() {
  ArduinoOTA.handle();
  EnviaEstadoOutputMQTT();
  client.loop();
  delay(3000);
}
```

Imagem 24. Recebimento e envio de respostas MQTT na função loop da NodeMCU.

IHM Web - API

```
//CONFIGURAÇÃO SERVIDOR HTTP
const server = express();
server.use(
  cors({
    origin: "*",
    methods: "GET,HEAD,PUT,PATCH,POST,DELETE",
    preflightContinue: false,
    optionsSuccessStatus: 204,
    credentials: true,
  })
);
server.use(bodyParser.urlencoded({ extended: false }));
server.use(bodyParser.json());
const PORT = 3000;
```

```
// CONFIGURAÇÕES BROKER LABORATÓRIO LEDS
const host = "mqtt://10.0.0.101";
const options = {
  port: 1883,
  clientId: "mqtt_pbl3SD",
  username: "aluno",
  password: "@luno*123",
};

const client = mqtt.connect(host, options);
```

Imagem 25. Configuração dos servidores HTTP e MQTT

IHM Web - API

```
//Escreve na Base de dados (arquivo Json)
const WriteFileSensors = async (topic, message) => {
  if (topic == topic_history0) {
    const path = "./data_bases/data_base_a0.json";
    WriteFile(path, message);
  } else if (topic == topic_historyD0) {
    const path = "./data_bases/data_base_s0.json";
    WriteFile(path, message);
  } else if (topic == topic_historyD1) {
    const path = "./data_bases/data_base_s1.json";
    WriteFile(path, message);
  } else if (topic == topic_historyD2) {
    const path = "./data_bases/data_base_s2.json";
    WriteFile(path, message);
  } else if (topic == topic_historyD3) {
    const path = "./data_bases/data_base_s3.json";
    WriteFile(path, message);
  } else if (topic == topic_historyD4) {
    const path = "./data_bases/data_base_s4.json";
    WriteFile(path, message);
```

```
// Busca todos os dados
server.get("/sensores/:sensor", async (req, res) => {
  res.header("Access-Control-Allow-Origin", "*");
  const sensor = req.params.sensor;
  if (sensor == "a0") {
    const path = "./data_bases/data_base_a0.json";
    const sensorData = await ReadFileSensors(path);
    return res.send(sensorData);
  } else if (sensor == "d0") {
    const path = "./data_bases/data_base_s0.json";
    const sensorData = await ReadFileSensors(path);
    return res.send(sensorData);
  } else if (sensor == "d1") {
    const path = "./data_bases/data_base_s1.json";
    const sensorData = await ReadFileSensors(path);
    return res.send(sensorData);
  } else if (sensor == "d2") {
    const path = "./data_bases/data_base_s2.json";
    const sensorData = await ReadFileSensors(path);
    return res.send(sensorData);
  } else if (sensor == "d3") {
    const path = "./data_bases/data_base_s3.json";
    const sensorData = await ReadFileSensors(path);
    return res.send(sensorData);
```

Imagem 26. Escrita e leitura dos dados na base de dados

IHM Web - Interface HTML

```
axios
  .get("http://localhost:3000/sensores/" + path)
  .then((response) => {
    dados = response.data;
    const json_values = dados[0];
    historico = json_values.historico;
    timestamps = formataData(json_values.timestamps);
```

```
if (chartGraph != null) {
  chartGraph.destroy();
}
chartGraph = new Chart(ctx, {
  type: "line",
  data: {
    labels: timestamps,
    datasets: [
      {
        label: name_sensor,
        backgroundColor: "rgba(0,0,0,1.0)",
        borderColor: "rgba(0,0,0,0.5)",
        data: historico,
      },
    ],
  },
  options: {
    scales: {
      y: {
        beginAtZero: true,
      },
    },
    plugins: {
      title: {
        display: true,
        text: "Sensores",
        padding: {
          top: 10,
          bottom: 30,
        },
      },
    },
```

Imagem 27. Requisição para API e montagem do gráfico

IHM Web - Interface HTML

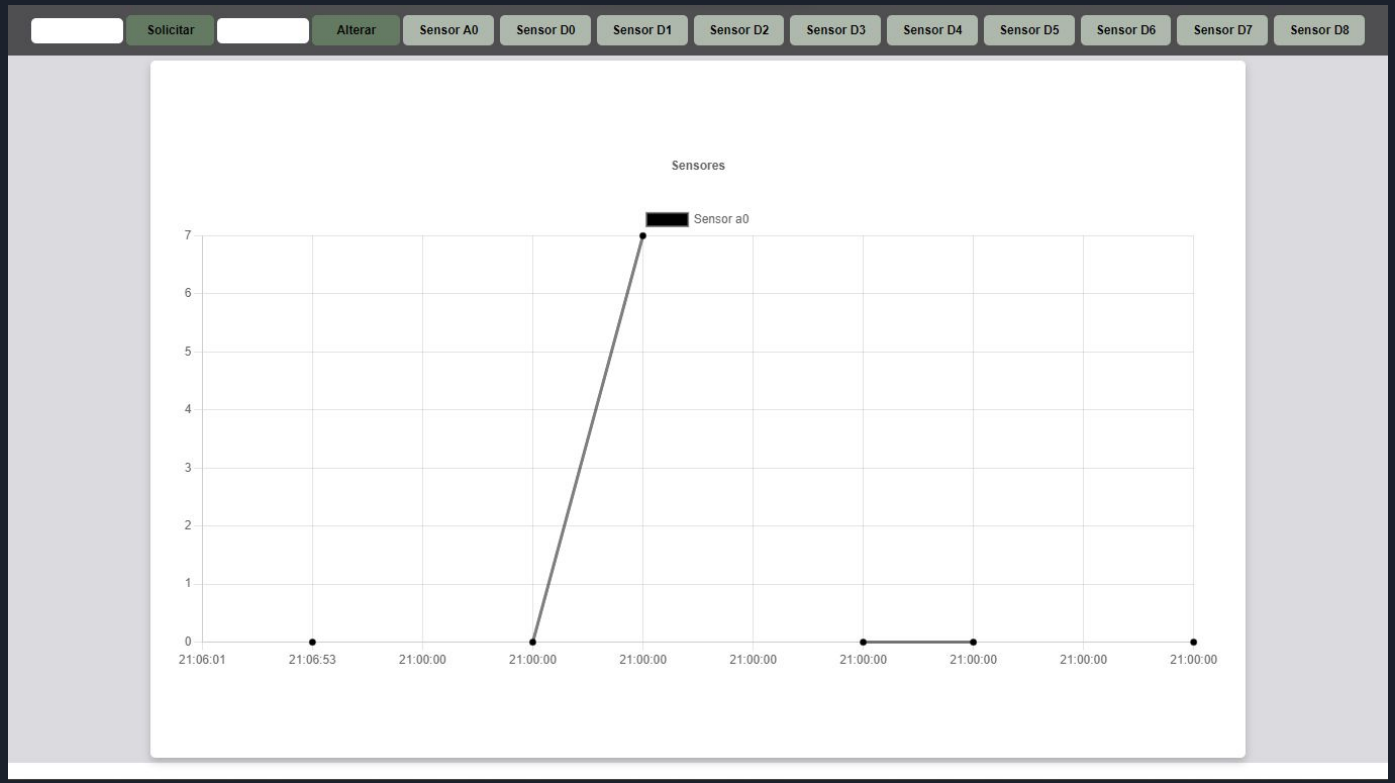


Imagem 28. Interface Web

Referências

Imagens 1, 3 a 28. Disponíveis em: https://github.com/ian-zaque/pbl_SD_3

Imagem 2. Orange Pi. Disponivel em: www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-PC-Plus.htm

Dados sobre Orange Pi PC Plus. Disponível em: www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-PC-Plus.html

Dados sobre NodeMCU ESP8266. Disponível em: www.filipeflop.com/produto/modulo-wifi-esp8266-nodemcu-esp-12