



PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
ENGENHARIA DA COMPUTAÇÃO

Ramon Vinícius Sila Corrêa

TRABALHO INTERDISCIPLINAR:
Automação Residencial

Belo Horizonte – MG

2021

1. INTRODUÇÃO

No mundo de hoje, a conectividade tem se tornado cada vez mais importante para as pessoas. A ideia de conectar vários equipamentos presentes em suas casas cresce cada vez mais com a vontade das pessoas de facilitar algumas tarefas do dia-a-dia, deixando tudo de forma automática.

O principal responsável por permitir toda essa tecnologia para as pessoas é a Internet das Coisas (IOT), que está cada vez mais popular fazendo com que o custo dela se torne menor.

Com a maior acessibilidade dessas tecnologias, a implementação delas para automação residencial é cada vez mais comum. Visando isso, este projeto mostra o desenvolvimento de um circuito de automação residencial utilizando ferramentas IOT.

2. OBJETIVOS

O objetivo central deste projeto é realizar um sistema de automação residencial baseado em ferramentas IOT para que as pessoas possam adaptá-los em suas casas.

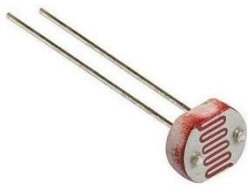

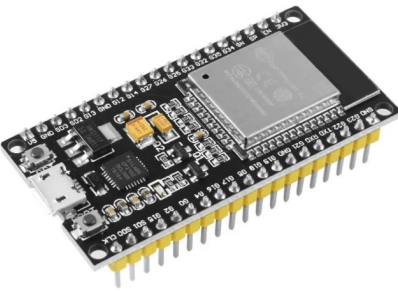
Desta forma, para alcançarmos esse objetivo iremos criar um sistema que tenha as seguintes funcionalidades:

- Controle de Acesso: O sistema de controle de acesso será feito por um sensor RFID que irá autorizar a entrada de pessoas através de um cartão magnético, além de conseguir realizar a liberação de acesso através de um app no celular;
- Controle de Temperatura: O sistema de controle de temperatura será realizado através de um sensor de temperatura que irá indicar no app do smartphone a temperatura atual e se passar de certa temperatura irá ativar um ventilador;
- Controle de Luminosidade: O Controle de luminosidade será feito com um LDR, que irá acender ou apagar a lâmpada através do nível de luz que estiver no ambiente, podendo também apagar ou acender a lâmpada através do celular.

3. PROJETO DE HARDWARE

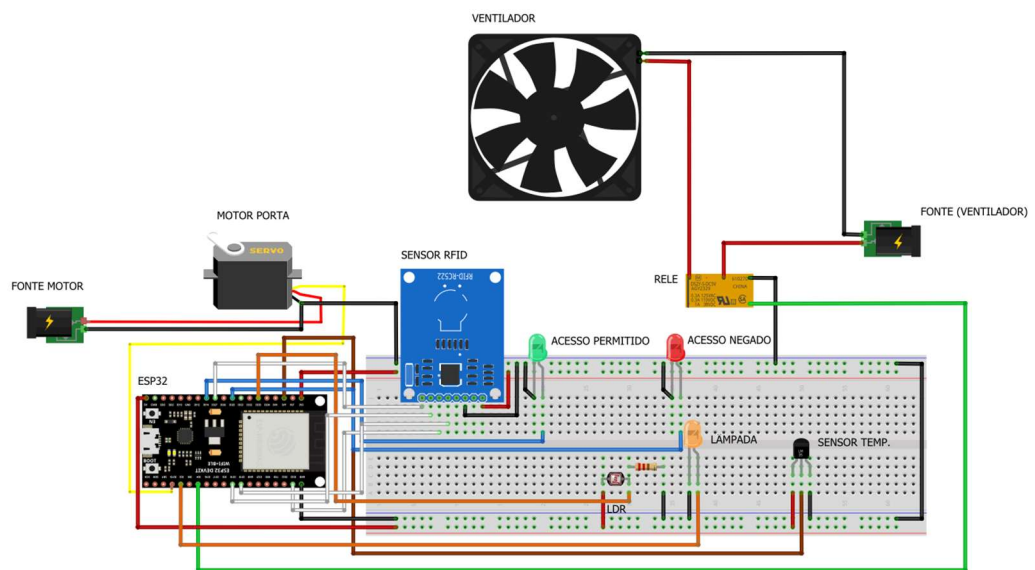
O projeto de Hardware é relativamente simples, ele utiliza o ESP32 para controlar o circuito, alguns sensores e atuadores, e outros componentes. Abaixo temos uma tabela com os principais componentes utilizados e sua funcionalidade no circuito:

COMPONENTE	FUNÇÃO
 SENSOR RFID	Este sensor é um leitor de crachá, que irá verificar se o crachá que a pessoa está tentando passar é válido ou não.
 SERVO MOTOR	O servo motor irá simular a tranca de uma porta, que irá abrir ou fechar com a validação do sensor RFID.
 LED's Verde e Vermelho	Os LED's verde e vermelho, irão indicar se o cartão passado no RFID é valido ou não (Verde – Válido / Vermelho – Inválido).
 SENSOR DE TEMPERATURA (LM35)	O sensor LM35 irá detectar a temperatura que está no ambiente.
	O cooler irá simular um ventilador, que é ativado quando o sensor LM35 identificar temperatura alta.

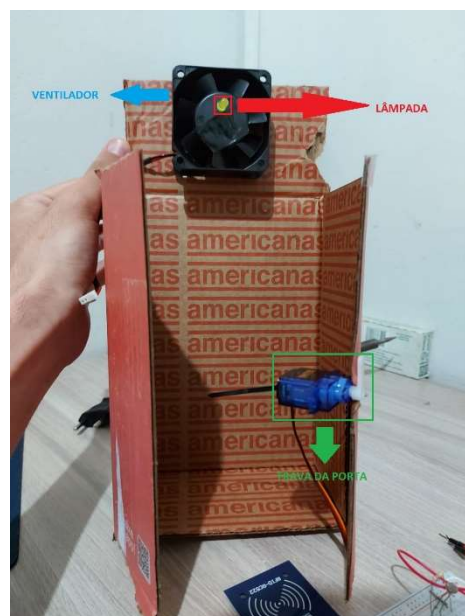
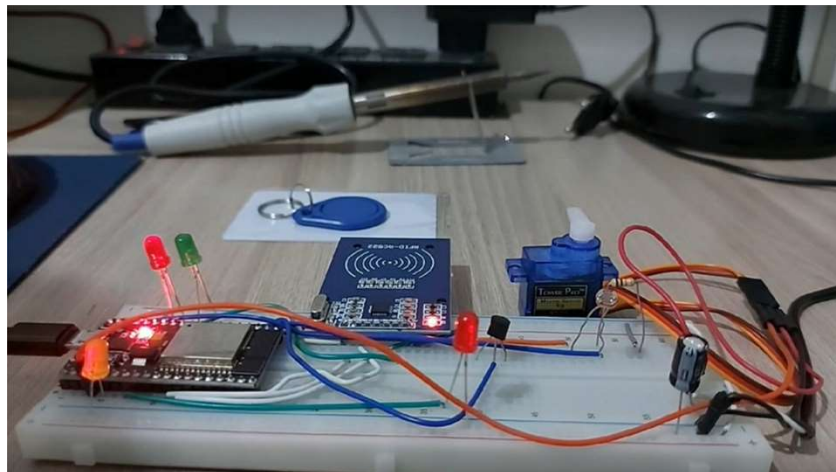
COOLER	
 <p>SENSOR DE LUMINOSIDADE (LDR)</p>	<p>O sensor LDR irá verificar a quantidade de luz no ambiente e irá acender uma lâmpada caso a luminosidade esteja baixa.</p>
 <p>LED AMARELO</p>	<p>O LED amarelo irá simular a lâmpada que é ativada pelo LDR.</p>
 <p>ESP32</p>	<p>O ESP32 será responsável por controlar todo o circuito, além de realizar a comunicação via Wi-Fi.</p>

3.1. DIAGRAMA ELÉTRICO

A seguir é mostrado o diagrama elétrico do circuito realizado no software Fritzing, onde mostra toda a ligação elétrica dos sensores e atuadores junto do ESP32:



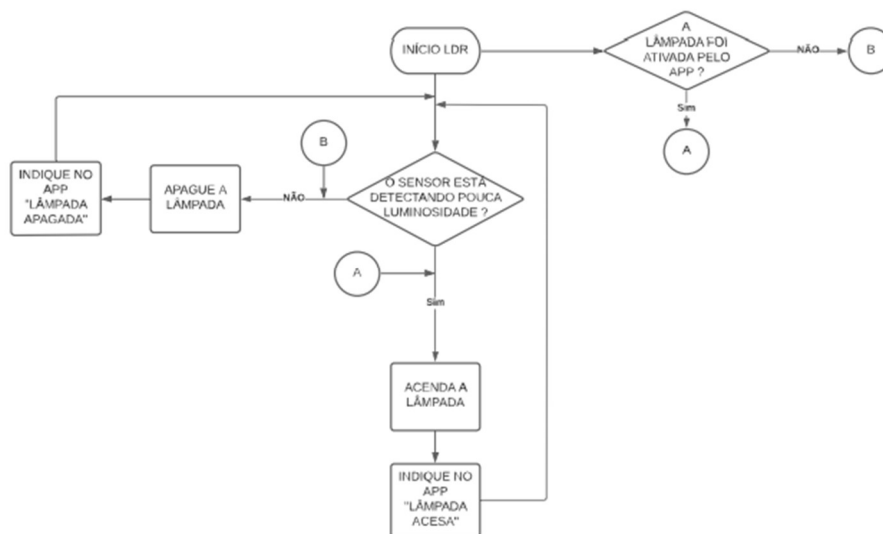
fritzing



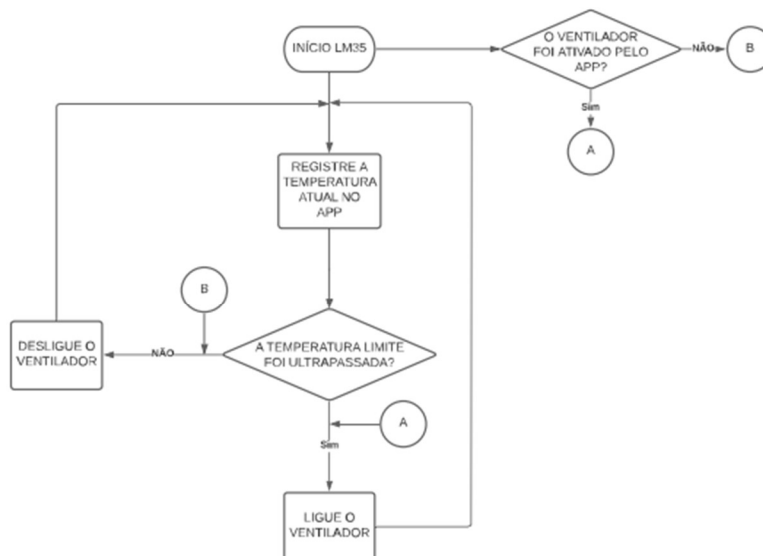
4. PROJETO DE SOFTWARE

Para este projeto foi desenvolvido o software para o ESP32 através da plataforma Arduino IDE, onde foi feita toda a programação dos sensores e atuadores. Além disso, foi elaborado um app através do MQTT DASH, que nos permite controlar os sensores e atuadores através do celular. A seguir você verá um fluxograma dividido em 3 partes, sendo essas partes o funcionamento de cada um dos sensores (RFID, Sensor de Temperatura, LDR):

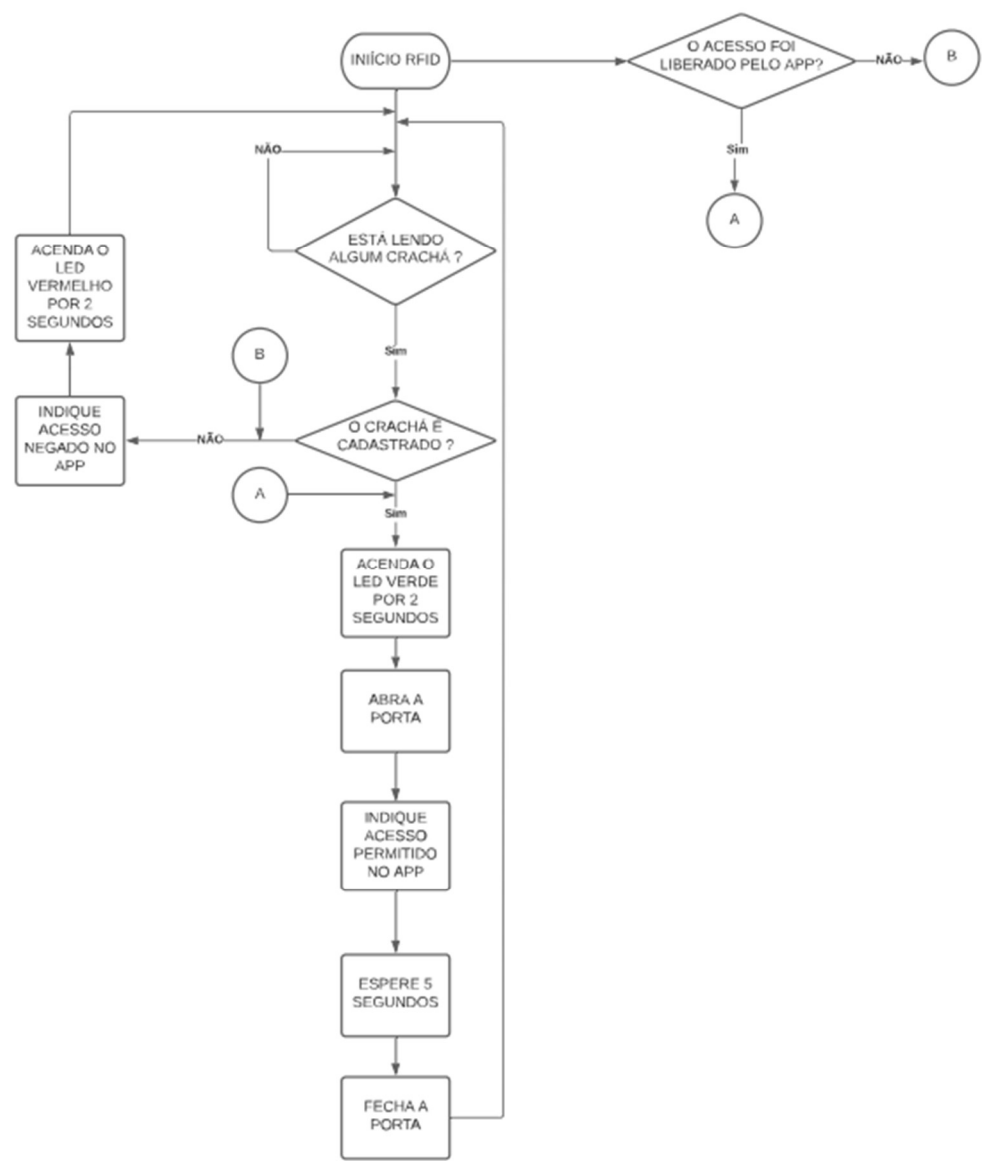
- SENSOR LDR



- SENSOR LM35



SENSOR RFID:



APP E FUNCIONAMENTO DO PRODUTO:

Para a montagem do app de smartphone, foi utilizado o MAQTT DASH, que permitiu criar uma interface simples para o controle e monitoramento dos sensores. A seguir, será mostrado alguns prints do app juntamente com uma explicação sobre o funcionamento do circuito:

- O sensor RFID irá ler um crachá e verificar se o mesmo é registrado ou não, caso ele seja registrado irá acender um LED verde permitindo o acesso da pessoa, caso não seja irá acender um LED vermelho negando o acesso. Essas informações de acesso permitido e negado são mostradas no MQTT DASH, sendo também possível liberar o acesso pelo app.



Figura 2: App indicando acesso liberado

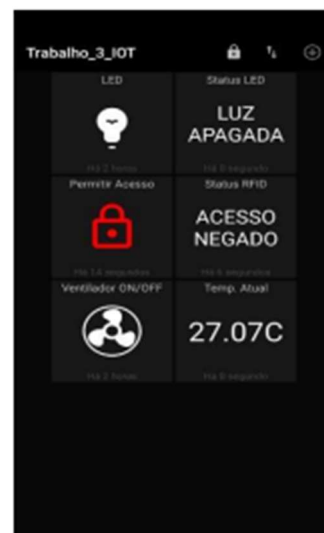


Figura 3: App indicando acesso negado

- O sensor LDR irá identificar a quantidade de luz de um ambiente e irá acender um LED amarelo caso a luminosidade esteja baixa e apagar caso esteja alta. O MQTT DASH irá mostrar o status do LED (Apagado ou Aceso), e você também poderá decidir se ele irá ficar ligado ou desligado apertando o botão indicado pelo desenho de uma lâmpada.



Figura 4: App indicando LED aceso

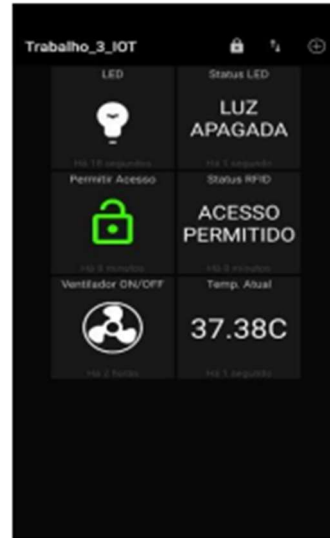


Figura 5: App indicando LED apagado

- O sensor de temperatura LM35 irá identificar a temperatura ambiente e irá ligar um cooler para refrigeração do local caso passe uma temperatura determinada (foi utilizado um LED vermelho apenas para simulação, futuramente será adicionado um cooler). O MQTT DASH irá mostrar a temperatura atual e te dará a opção de ligar ou desligar o cooler a qualquer momento através de um botão indicado pela figura de um ventilador.

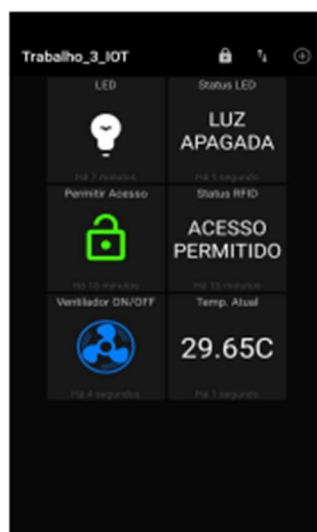


Figura 7: App indicando ventilador ligado e mostrando temperatura atual

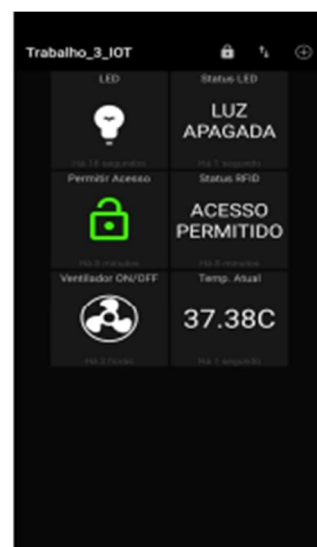


Figura 6: App indicando ventilador desligado e mostrando temperatura atual

LINGUAGEM UTILIZADA:

A linguagem de programação utilizada neste projeto foi a Linguagem C++, pois é a linguagem adotada pela IDE do Arduino, além de ser uma linguagem fácil de se compreender, facilitando o desenvolvimento do projeto.

Para o Banco de Dados foi utilizado a linguagem Python, armazenando os valores de temperatura obtidos pelo sensor.

Os códigos fontes do ESP32 e BD estão em anexo ao final deste relatório.

5. CUSTO DO PROJETO

A seguir será mostrada uma tabela com todos os componentes utilizados no projeto e seus respectivos preços no mercado:

COMPONENTE	PREÇO
Sensor RFID MFRC522	R\$ 26,90
Micro Servo 9g SG90 TowerPro	R\$ 24,90
LED Difuso 5mm (Vermelho, Verde, Amarelo)	R\$ 0,90 (R\$ 0,30 - 1 unid.)
Sensor de Temperatura LM35	R\$ 14,90
Cooler 12V	R\$ 19,90
Sensor de Luminosidade LDR	R\$ 0,90
Resistor 10K (10 Unidades)	R\$ 2,00
ESP32 38 Pinos	R\$ 79,90
Protoboard 830 pinos	R\$ 19,90
Fonte 12V 1A	R\$ 22,90
Módulo Relé	R\$ 8,90
TOTAL	R\$ 222,00

Ao todo, o projeto custou em torno de R\$ 222,00, podendo aumentar o custo se ele for realmente implementado em uma residência.

6. CONCLUSÃO

O IOT (Internet Of Things – Internet das Coisas), é uma área que tem crescido cada vez mais tendo em vista a necessidade das pessoas de se conectarem aos seus aparelhos. Sendo assim uma aplicação de IOT que é bastante comum é em automação residencial, sendo esse o tema abordado durante a execução deste projeto.

Com este projeto foi possível ver como funcionam algumas ferramentas IOT, além de ampliar nosso conhecimento em outras áreas envolvidas, como a programação, eletrônica e internet.

O tema deste projeto colaborou com o aprendizado da matéria sendo uma ótima maneira de demonstrar algumas aplicações interessantes do IOT.

ANEXO 1: Programa ESP32

```

/*****

-- TRABALHO INTERDISCIPLINAR: IOT --

ALUNO: Ramon Vinícius Silva Corrêa
TURMA: 3132.1.00

*****/

/*-- INCLUSÃO DE BIBLIOTECAS --*/

#include <WiFi.h> //Biblioteca para Wi-Fi
#include <PubSubClient.h> //Biblioteca para Pub e Sub
#include <SPI.h> //Biblioteca para comunicação SPI do sensor RFID
#include <MFRC522.h> //Biblioteca para sensor RFID
#include <Servo.h> //Biblioteca para Servo Motor
#include <Wire.h>

/*****

/*-- DEFINIÇÕES PARA SENSOR RFID --*/

#define ID "27 6E 6A C9" //Define ID que será aceito pelo sensor RFID
#define LedVerde 26 //Define pino para Led Verde de indicação de "ACESSO LIBERADO"
#define LedVermelho 12 //Define pino para Led vermelho de indicação de "ACESSO NEGADO"
#define SS_PIN 14
#define RST_PIN 27

/*****

/*-- DEFINIÇÕES PARA SENSOR LDR --*/

#define portaLDR 35 //Define pino para sensor LDR
#define PIN_LED 2 //Define pino para LED
#define LED_BUILTIN 2

/*****

/*-- DEFINIÇÕES PARA SENSOR DE TEMPERATURA --*/

#define ADC_VREF_mV 3300.0 // in millivolt
#define ADC_RESOLUTION 4096.0
#define PIN_LM35 A0 //define pino analógico para LM35
```

```
#define PIN_VENT    4 //Define pino para acionar o cooler
```

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

```
/*-- DEFINIÇÕES PARA MQTT --*/
```

```
#define TOPICO_SUBSCRIBE_LED    "topico_liga_desliga_led"
```

```
#define TOPICO_SUBSCRIBE_RFID    "topico_rfid"
```

```
#define TOPICO_SUBSCRIBE_VENT    "topico_vent"
```

```
#define TOPICO_PUBLISH_TEMPERATURA    "topico_sensor_temperatura"
```

```
#define TOPICO_PUBLISH_RFID    "topico_rfid"
```

```
#define TOPICO_PUBLISH_LDR    "topico_liga_desliga_led"
```

```
#define ID_MQTT    "Trabalho_3_IOT"    //id mqtt (para identificação de sessão)
```

```
const char* SSID    = "LIVE TIM_0820_2G"; // SSID / nome da rede WI-FI que deseja se conectar
```

```
const char* PASSWORD = "a3ehn6rep6"; // Senha da rede WI-FI que deseja se conectar
```

```
const char* BROKER_MQTT = "test.mosquitto.org";
```

```
int BROKER_PORT = 1883; // Porta do Broker MQTT
```

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

```
/*-- VARIÁVEIS E OBJETOS GLOBAIS --*/
```

```
WiFiClient espClient; // Cria o objeto espClient
```

```
PubSubClient MQTT(espClient); // Instancia o Cliente MQTT passando o objeto espClient
```

```
MFRC522 mfrc522(SS_PIN, RST_PIN); // define os pinos de controle do modulo de leitura de cartoes RFID
```

```
Servo myServo; //Cria objeto para servo motor
```

```
int j = 0, k=0; //Variáveis para flag
```

```
int adcVal = analogRead(PIN_LM35); //Variável para guardar valor do sensor de temperatura
```

```
int ativaServo = 0;
```

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

```
/*-- PROTOTYPES --*/
```

```
void initWiFi(void);
```

```
void initMQTT(void);
```

```
void mqtt_callback(char* topic, byte* payload, unsigned int length);
```

```
void reconnectMQTT(void);
```

```
void reconnectWiFi(void);
```

```
void VerificaConexoesWiFIEMQTT(void);
```

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

```
/*-- FUNÇÃO PARA CONECTAR-SE AO WI-FI --*/
```

```
void initWiFi(void)
```

```
{  
    delay(10); //Delay de 10 milissegundos  
    Serial.println("-----Conexao WI-FI-----"); //Imprime mensagem no monitor serial  
    Serial.print("Conectando-se na rede: "); //Imprime mensagem no monitor serial  
    Serial.println(SSID); //Imprime mensagem no monitor serial  
    Serial.println("Aguarde"); //Imprime mensagem no monitor serial  
    reconnectWiFi(); //Chama função para reconectar o Wi-Fi  
}
```

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

```
/*-- FUNÇÃO PARA CONECTAR-SE AO MQTT --*/
```

```
void initMQTT(void)
```

```
{  
    MQTT.setServer(BROKER_MQTT, BROKER_PORT); //informa qual broker e porta deve ser conectado  
    MQTT.setCallback(mqtt_callback); //atribui função de callback (função chamada quando qualquer informação de um dos tópicos subscritos chega)  
}
```

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

```
/*-- FUNÇÃO DE CALLBACK--*/
```

```
//esta função é chamada toda vez que uma informação de um dos tópicos subscritos chega
```

```
void mqtt_callback(char* topic, byte* payload, unsigned int length)
```

```
{  
    char str_LDR[20] = {0};  
    String msg;  
  
    /* obtem a string do payload recebido */  
    for (int i = 0; i < length; i++)  
    {  
        char c = (char)payload[i];
```

```

    msg += c;
}

Serial.print("Chegou a seguinte string via MQTT: ");
Serial.println(msg);

/* toma ação dependendo da string recebida */

/*-----*/
//Se chegar a mensagem L acende um led e publica no MQTT
if (msg.equals("L"))
{
    digitalWrite(PIN_LED, HIGH);
    Serial.println("LED aceso mediante comando MQTT");
    sprintf(str_LDR, "LUZ ACESA");
    MQTT.publish(TOPICO_PUBLISH_LDR, str_LDR);
    j = 1;
}

//Se chegar a mensagem D desliga o led e publica no MQTT
if (msg.equals("D"))
{
    digitalWrite(PIN_LED, LOW);
    Serial.println("LED apagado mediante comando MQTT");
    sprintf(str_LDR, "LUZ APAGADA");
    MQTT.publish(TOPICO_PUBLISH_LDR, str_LDR);
    j=0;
}
/*-----*/

//Se chegar a mensagem 1 liga o cooler
if (msg.equals("1"))
{
    digitalWrite(PIN_VENT, HIGH);
    k = 1;
}

//Se chegar a mensagem 0 desliga o cooler

```

```
    if (msg.equals("0"))
    {
        digitalWrite(PIN_VENT, LOW);
        k = 0;
    }

/*-----*/

//Se chegar a mensagem P liga o Led verde por 2seg
if(msg.equals("P"))
{
    digitalWrite(LedVerde, HIGH);
    delay(2000);
    digitalWrite(LedVerde, LOW);

    ativaServo = 1;

    if(ativaServo == 1)
    {
        for(int pos = 0; pos < 90; pos++)
            myServo.write(pos);

        delay(3000);

        for(int pos = 90; pos > 0; pos--)
            myServo.write(pos);

        ativaServo = 0;
    }
}

//Se chegar a mensagem N liga o led vermelho por 2seg
if(msg.equals("N"))
{
    digitalWrite(LedVermelho, HIGH);
    delay(2000);
    digitalWrite(LedVermelho, LOW);
```



```
}  
}
```

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

```
/*-- FUNÇÃO DE RECONECTAR-SE AO BROKER MQTT--*/
```

```
void reconnectMQTT(void)  
{  
  while (!MQTT.connected())  
  {  
    Serial.print("* Tentando se conectar ao Broker MQTT: ");  
    Serial.println(BROKER_MQTT);  
    if (MQTT.connect(ID_MQTT))  
    {  
      Serial.println("Conectado com sucesso ao broker MQTT!");  
      MQTT.subscribe(TOPICO_SUBSCRIBE_LED);  
      MQTT.subscribe(TOPICO_SUBSCRIBE_RFID);  
      MQTT.subscribe(TOPICO_SUBSCRIBE_VENT);  
    }  
    else  
    {  
      Serial.println("Falha ao reconectar no broker.");  
      Serial.println("Havera nova tentatica de conexao em 2s");  
      delay(2000);  
    }  
  }  
}
```

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

```
/*-- FUNÇÃO DE VERIFICAR CONEXÃO WI-FI E MQTT--*/
```

```
void VerificaConexoesWiFiEMQTT(void)  
{  
  if (!MQTT.connected())  
    reconnectMQTT(); //se não há conexão com o Broker, a conexão é refeita  
  reconnectWiFi(); //se não há conexão com o WiFI, a conexão é refeita  
}
```

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

```
/*-- FUNÇÃO PARA RECONECTAR-SE AO WI-FI--*/
```

```
void reconnectWiFi(void)
{
    //se já está conectado a rede WI-FI, nada é feito.
    //Caso contrário, são efetuadas tentativas de conexão
    if (WiFi.status() == WL_CONNECTED)
        return;
    WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }
    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(SSID);
    Serial.println("\nIP obtido: ");
    Serial.println(WiFi.localIP());
}
```

```
/*-----*/
```

```
/*-- FUNÇÃO PARA SENSOR RFID--*/
```

```
void sensorRFID()
{
    char rfid_str[20] = {0};

    if ( ! mfr522.PICC_IsNewCardPresent()) {
        return;          // se nao tiver um cartao para ser lido recomeça o void loop
    }
    if ( ! mfr522.PICC_ReadCardSerial()) {
        return;          //se nao conseguir ler o cartao recomeça o void loop tambem
    }

    String conteudo = "";    // cria uma string

    Serial.print("id da tag :"); //imprime na serial o id do cartao
```

```

for (byte i = 0; i < mfrc522.uid.size; i++)
{
    //faz uma verificacao dos bits da memoria do cartao

    //ambos comandos abaixo vão concatenar as informacoes do cartao...

    //porem os 2 primeiros irao mostrar na serial e os 2 ultimos guardaraos os valores na string de conteudo para fazer as
    verificacoes

    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
    conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    conteudo.concat(String(mfrc522.uid.uidByte[i], HEX));
}

Serial.println();

conteudo.toUpperCase(); // deixa as letras da string todas maiusculas

if (conteudo.substring(1) == ID)// verifica se o ID do cartao lido tem o mesmo ID do cartao que queremos liberar o acesso
{
    ativaServo = 1;
    digitalWrite(LedVerde, HIGH); //liga o led verde
    sprintf(rfid_str, "ACESSO PERMITIDO"); //Imprime no MQTT uma mensagem
    delay(2000); //Aguarda 2 segundos
    digitalWrite(LedVerde, LOW); //desliga o led verde

    MQTT.publish(TOPICO_PUBLISH_RFID, rfid_str);

}

else // caso o cartao lido nao foi registrado
{

    digitalWrite(LedVermelho, HIGH); // liga o led vermelho
    sprintf(rfid_str, "ACESSO NEGADO"); //Imprime uma mensagem no MQTT
    delay(2000); //Aguarda 2 segundos
    digitalWrite(LedVermelho, LOW); // desliga o led vermelho
    MQTT.publish(TOPICO_PUBLISH_RFID, rfid_str);
}

```

```

if(ativaServo == 1)
{
    for(int pos = 0; pos < 90; pos++)
        myServo.write(pos);

    delay(3000);

    for(int pos = 90; pos > 0; pos--)
        myServo.write(pos);

    ativaServo = 0;
}
}

/*****
                                     /*-- FUNÇÃO PARA SENSOR LDR --*/

void leituraLDR()
{
    char str_LDR[20] = {0};
    int leitura = analogRead(portaLDR); // realizar leitura do sensor ldr

    if((analogRead(portaLDR) < 1000)&&(j==0)) //Verifica Luminosidade do ambiente
    {
        //Se a luminosidade estiver baixa acende o led
        digitalWrite(PIN_LED, HIGH);
        sprintf(str_LDR, "LUZ ACESA");
        MQTT.publish(TOPICO_PUBLISH_LDR, str_LDR);
    }

    else
    {
        //Se a luminosidade estiver alta deixa o led apagado
        digitalWrite(PIN_LED, LOW);
        sprintf(str_LDR, "LUZ APAGADA");
        MQTT.publish(TOPICO_PUBLISH_LDR, str_LDR);
    }
}

```

```

/*****

                                /*-- INICIA O SETUP --*/

void setup()
{
    myServo.attach(15); //Define pino do servo
    myServo.write(0); //inicia o Servo motor em posição 0°

    Serial.begin(115200); // inicia a comunicacao serial com o computador na velocidade de 115200 baud rate
    Serial.println("Disciplina IoT: acesso a nuvem via ESP32");

    pinMode(PIN_LED, OUTPUT); // programa LED interno como saida
    digitalWrite(PIN_LED, LOW); // inicia o LED apagado

    initWiFi(); //Inicializa a conexao wi-fi
    initMQTT(); //Inicializa a conexao ao broker MQTT

    SPI.begin(); // inicia a comunicacao SPI que sera usada para comunicacao com o modulo RFID
    mfrc522.PCD_Init(); //inicia o modulo RFID
    Serial.println("RFID + ESP32");
    Serial.println("Aguardando tag RFID para verificar o id da mesma.");
    pinMode(LedVerde, OUTPUT); //Define led verde como saida
    pinMode(LedVermelho, OUTPUT); //Define led vermelho como saida

    pinMode(PIN_VENT, OUTPUT); //Define pino do cooler como saida
}

*****/

                                /*-- INICIA O LOOP --*/

void loop()
{
    /* garante funcionamento das conexões WiFi e ao broker MQTT */
    VerificaConexoesWiFiMQTT();

    /*-----*/

    /* -- COMANDOS PARA LM35 --*/

```

```

int adcVal = analogRead(PIN_LM35); // Obtem o vaor ADC do sensor de temperatura

float milliVolt = adcVal * (ADC_VREF_mV / ADC_RESOLUTION); //Converte o valor obtido em volts para milivolts

float tempC = milliVolt / 10; //Converte o valor em volts para temperatura em graus celsius

char temperatura_str[10] = {0}; // cria string para temperatura

sprintf(temperatura_str, "%.2fC", tempC); // formata a temperatura aleatoria como string

// Publica a temperatura
MQTT.publish(TOPICO_PUBLISH_TEMPERATURA, temperatura_str);
Serial.print("Gerando temperatura aleatoria: ");
Serial.println(tempC);

if(k==0)
{
    //Se o sensor detectar um valor maior que 50°C liga o cooler
    if(tempC > 50.00)
        digitalWrite(PIN_VENT, HIGH);

    else
        digitalWrite(PIN_VENT, LOW);
}

/*-----*/

sensorRFID(); //Chama função para sensor RFID

if(j==0)
    leituraLDR(); //Chama função para LDR

MQTT.loop(); // keep-alive da comunicação com broker MQTT
delay(2000); //Refaz o ciclo após 2 segundos
}

```

ANEXO 2: Programação BD

```
import paho.mqtt.client as mqtt

# importe o conector do Python com o MySQL: instalar novamente neste env
import mysql.connector

# agora é necessário criar um objeto de conexão: encontra o MySQL e informa as credenciais para se conectar ao banco

# instalar novamente o conector: pip install mysql-connector-python
con = mysql.connector.connect(host='localhost',database='IoT',user='root',password='m0n.sql')

# verifique se a conexão ao BD foi realizada com sucesso
if con.is_connected():

    db_info = con.get_server_info()

    print("Conectado com sucesso ao Servidor ", db_info)

    # a partir de agora pode-se executar comandos SQL: para tanto é necessário criar um objeto tipo cursor
    # o cursor permite acesso aos elementos do BD
    cursor = con.cursor()

    # agora você pode executar o seu comando SQL. Por exemplo o comando "select database();" mostra o BD atual selecionado
    cursor.execute("select database();")

    # crio uma variável qualquer para receber o retorno do comando de execução
    linha = cursor.fetchone()

    print("Conectado ao DB", linha)

    # createTable é usada no comando SQL para criar a tabela dadosIoT, que só tem um registro chamado "mensagem"
    createTable = """

        CREATE TABLE dadosIoT (id INT AUTO_INCREMENT,

                                mensagem TEXT(512),

                                PRIMARY KEY (id));

    """

    # este par try/except verifica se a tabela já está criada. Se a tabela não existe, cai no try e é criada
    # se existe, cai no except e só mostra a mensagem que a tabela existe
    try:

        cursor.execute(createTable)

    except:

        print("Tabela dadosIoT já existe.")

        pass

def print_hi(name):

    # mensagem inicial

    print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.

# esta função é a função callback informando que o cliente se conectou ao servidor
```

```

def on_connect(client, userdata, flags, rc):
    print("Connectado com código "+str(rc))
    # assim que conecta, assina um tópico. Se a conexão for perdida, assim que
    # que reconectar, as assinaturas serão renovadas
    client.subscribe("topico_sensor_temperatura")
# esta função é a função callback que é chamada quando uma publicação é recebida do servidor
def on_message(client, userdata, msg):
    print("Mensagem recebida no tópico: " + msg.topic)
    print("Mensagem: "+ str(msg.payload.decode()) + "")
    # ao receber um dado, insere como um registro da tabela dadosIoT
    cursor = con.cursor()
    cursor.execute("INSERT INTO dadosIoT (mensagem) VALUES('{}')".format(str(msg.payload.decode())))
    con.commit()
    cursor.execute("SELECT * FROM dadosIoT")
    myresult = cursor.fetchall()
    print(myresult)
    if str(msg.payload.decode().strip()) == "termina":
        print("Recebeu comando termina.")
    if con.is_connected():
        cursor.close()
    con.close()
    print("Fim da conexão com o Banco dadosIoT")
    if str(msg.payload.decode().strip()) == "delete":
        cursor.execute("TRUNCATE TABLE dadosIoT")
    if __name__ == '__main__':
        print_hi('Olá Turma.')
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message
    client.connect("test.mosquitto.org", 1883, 60)
    # client.connect("broker.hivemq.com", 1883, 60) # broker alternativo
    # a função abaixo manipula tráfego de rede, trata callbacks e manipula reconexões.
    client.loop_forever()

```