

**Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
SSC0952 - Internet das Coisas**

**Sistema de iluminação com
controle remoto**

Gabriel Pontes - 9313030
Jade Bortot de Paiva- 11372883
João Pedro D. Torrezan - 9806933
Augusto Freitas - 8937191

Prof. Dr. Julio Cezar Estrella

**São Carlos - SP
2019**

Resumo

O projeto desenvolvido é uma abordagem para o controle remoto do funcionamento de sistemas de iluminação. A solução visa uma economia de energia e maior praticidade no controle das lâmpadas. Essa automação utiliza alguns dispositivos, como o microcontrolador ESP32, alguns relés, que funcionam como interruptores, sensores de presença, pode, ou não, contar com uma Raspberry para o controle central da aplicação, e por fim a presença de uma rede sem fio no local da aplicação. Todo o controle do sistema é feito de forma remota, através da rede local, ou ainda pode contar com acesso externo. Os detalhes do funcionamento e montagem do projeto serão descritos no decorrer do relatório.

Link do projeto

https://github.com/augustofreitas4/trab_iot_grupo5

Introdução

Finalidade e objetivos

Com a finalidade de reduzir o consumo e gerar mais comodidade, foi desenvolvido um sistema de iluminação automatizada que pode ser baseada no controle de presença, utilizando de sensores ou então um controle convencional, de acendimento ou desligamento das lâmpadas.

Desenvolvimento

Arquitetura

A arquitetura do sistema foi desenvolvida baseada nas ideias iniciais e nas ferramentas disponíveis. O sistema é dividido em 2 componentes principais, o microcontrolador ESP32 e o servidor, podendo ser hospedado na Raspberry Pi, ou ainda em Clouds.

ESP32

O microcontrolador fica responsável pelo controle dos relés e administração da informação recebida pelo sensor de presença. Como este microcontrolador possui acesso ao Wifi, conseguimos fazer o envio e recebimento de dados, para que, através disso, possamos administrar remotamente o funcionamento das luzes, e obtermos dados do sensor. Estabelecida a comunicação o funcionamento do ESP é independente do restante do projeto.

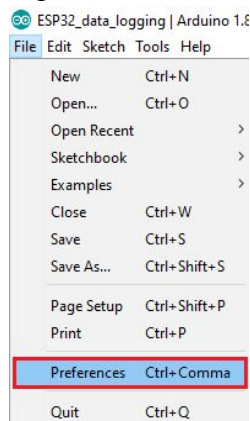


Figura 1: Menu File do Arduino IDE

Sendo que primeiro foi necessário instalar o ESP32 no Arduino IDE para poder programá-lo.

Com a IDE aberta, clique em File > Preferencias (Arquivo > Preferências)

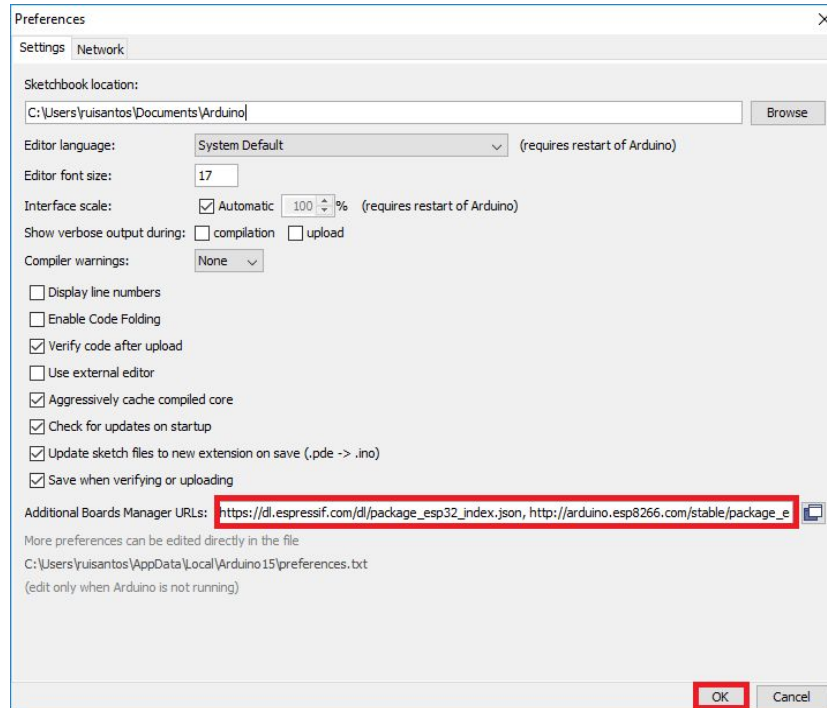


Figura 2: Janela aberta ao clicar em Preferências

Adicionar no campo “Additional Boards Manager URLs” a seguinte URL:
https://dl.espressif.com/dl/package_esp32_index.json
após clique em OK.

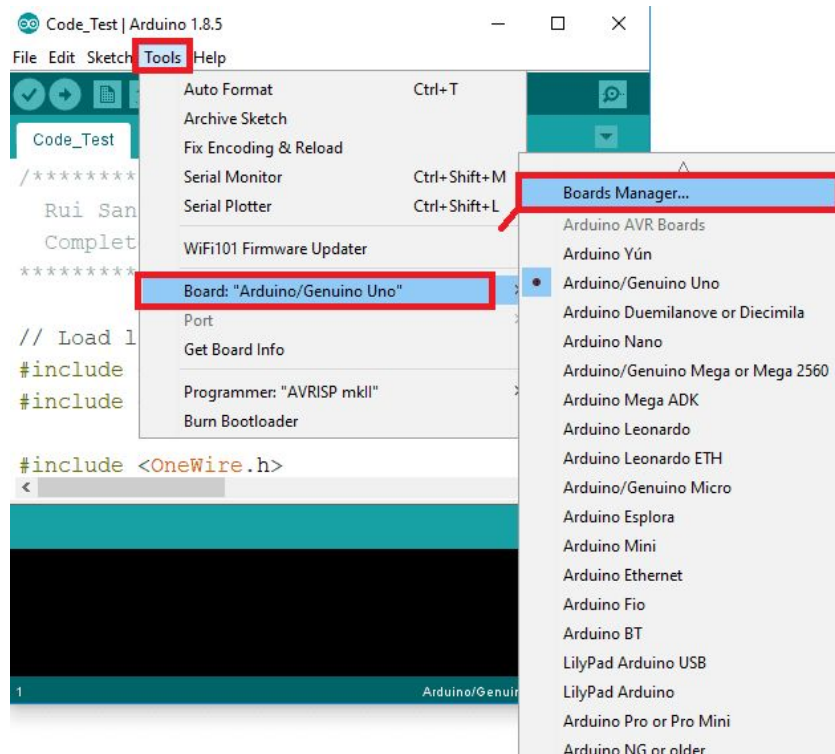


Figura 3: Menu Tools do Arduino IDE

Abra o gerenciador de placas clicando em Tools > Board > Boards Manager (Ferramentas > Placa “xxxxx” > Gerenciador de Placas).

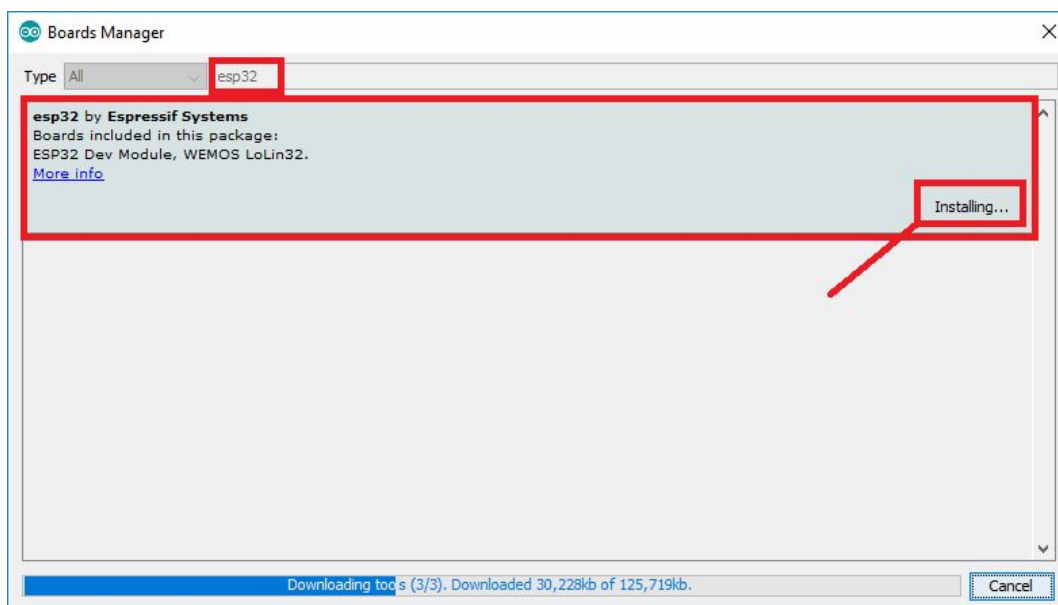


Figura 4: Boards Manager do Arduino IDE durante a instalação do ESP32

Procure por ‘esp32’ na caixa de busca. O resultado ‘esp32 by Espressif System’ deve aparecer. Clique no botão install (instalar) nesta opção.

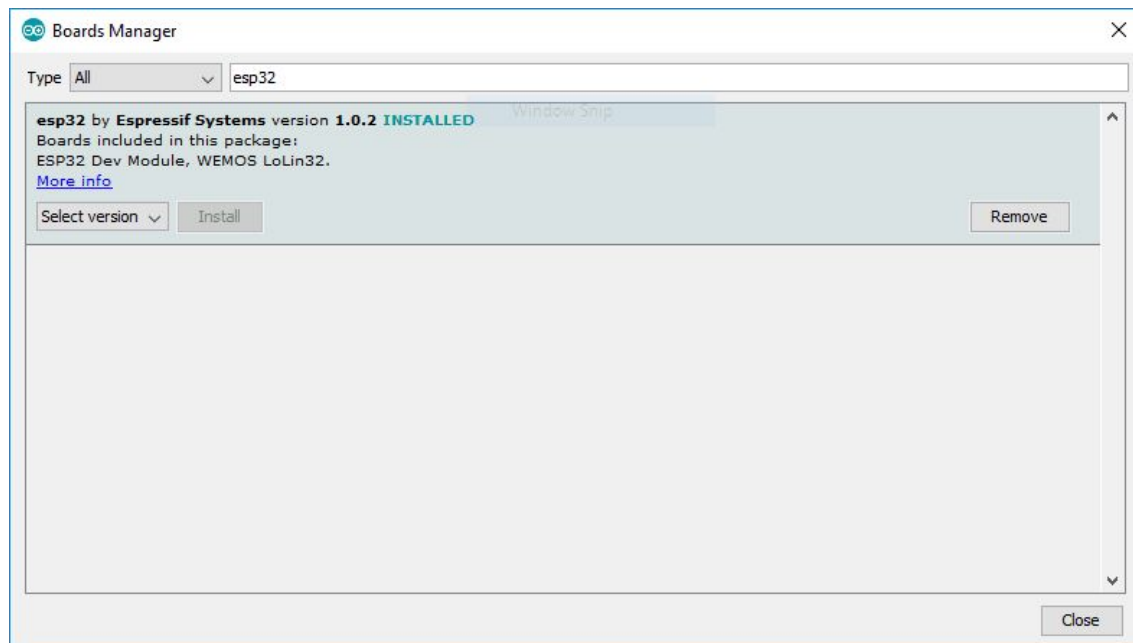


Figura 5: Boards Manager do Arduino IDE depois da instalação do ESP32

Se tudo ocorrer corretamente até aqui, deve-se ver a tela acima

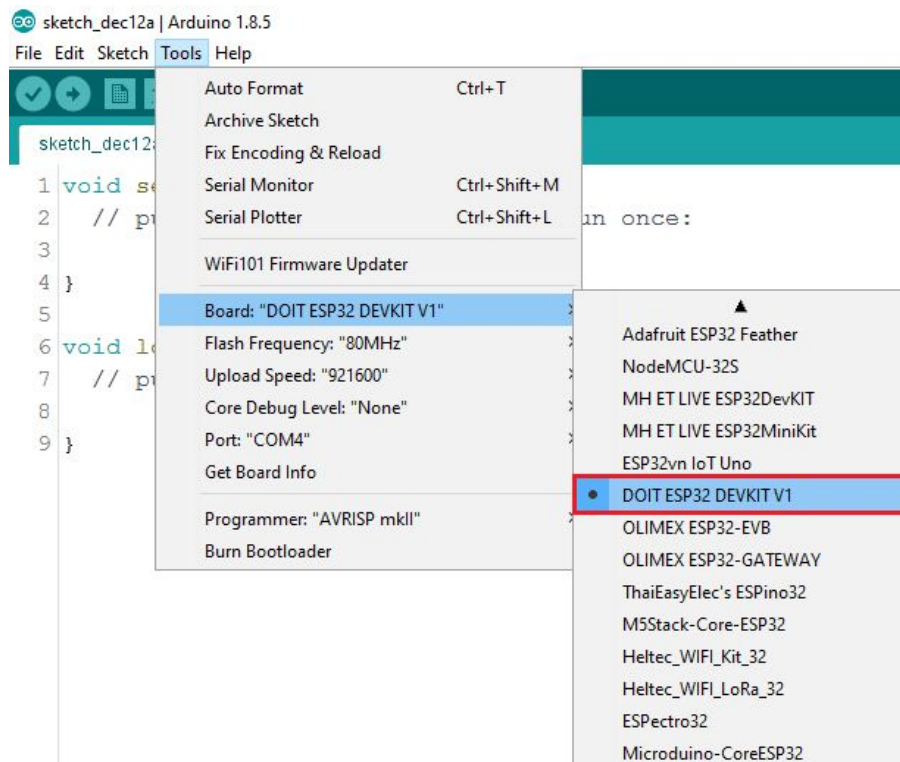


Figura 6: Encontrando a placa do ESP32 nas placas registradas

Inicialmente é necessário selecionar o compilador da ESP que foi instalado. Para isso clique em Tools > Board (Ferramentas > Placa) e selecione a opção DOIT ESP32 DEVKIT V1

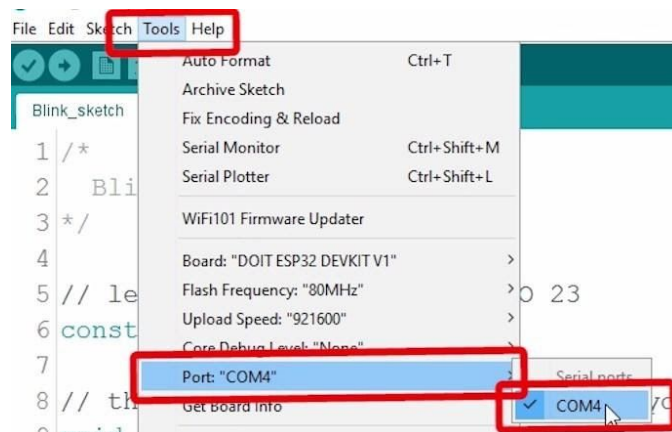


Figura 7: Selecionando a porta que está o ESP32

Em seguida, é necessário selecionar a porta serial em que a ESP32 foi conectada ao computador (respectiva porta usb). É necessário que os drivers de conexão usb/serial estejam instalados corretamente para que esse passo funcione (é possível encontrá-los neste [link](#)). Para selecionar a porta serial basta clicar em Tooles > Port > COMx (Ferramentas > Porta > COMx), onde x indica o número da porta.

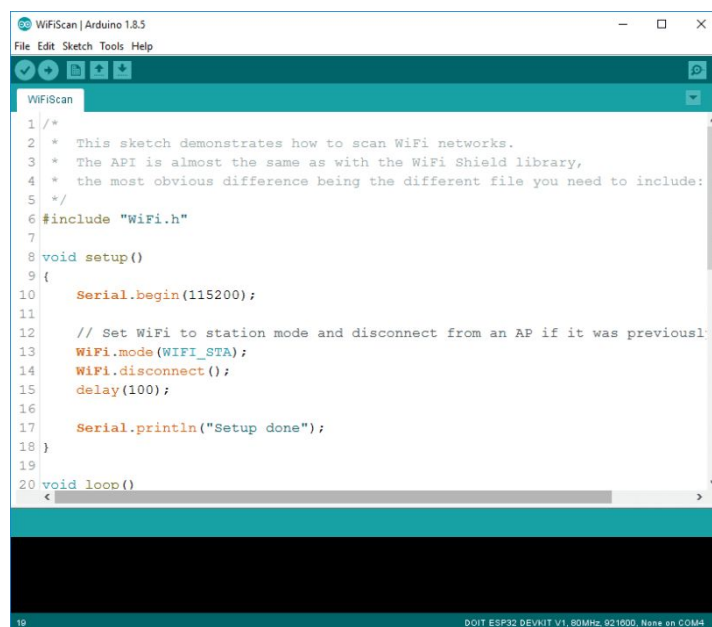
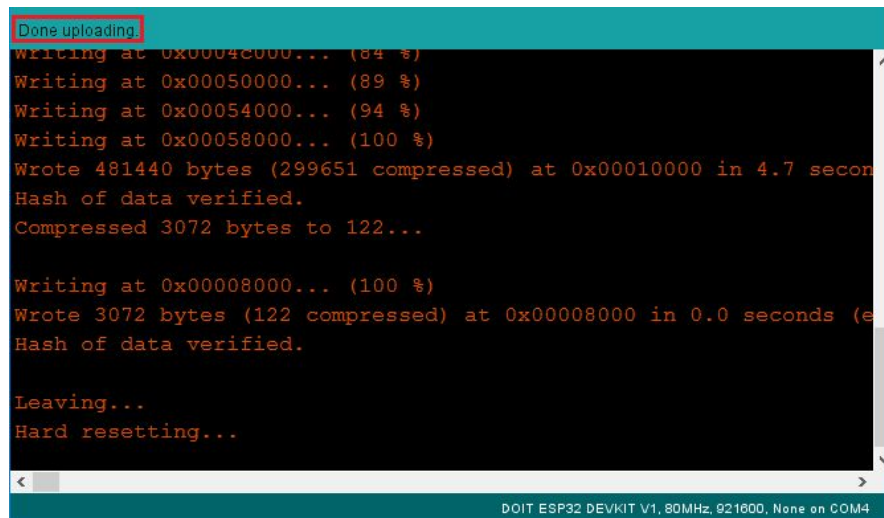


Figura 8: Código teste do ESP32

A biblioteca adiciona vem com muitos códigos “pré-prontos” que podem ser utilizados como casos de teste. Vamos usar como teste um código que detecta as redes wifi disponíveis no alcance da ESP32 e envia por comunicação serial ao computador para que possamos visualizá-las no IDE do Arduino. Para abrir esse teste clique em File > Examples > WiFi

(divisória da ESP) > WiFiScan (Arquivo > Exemplos > WiFi > WiFiScan). Uma nova janela será aberta com o código de teste.



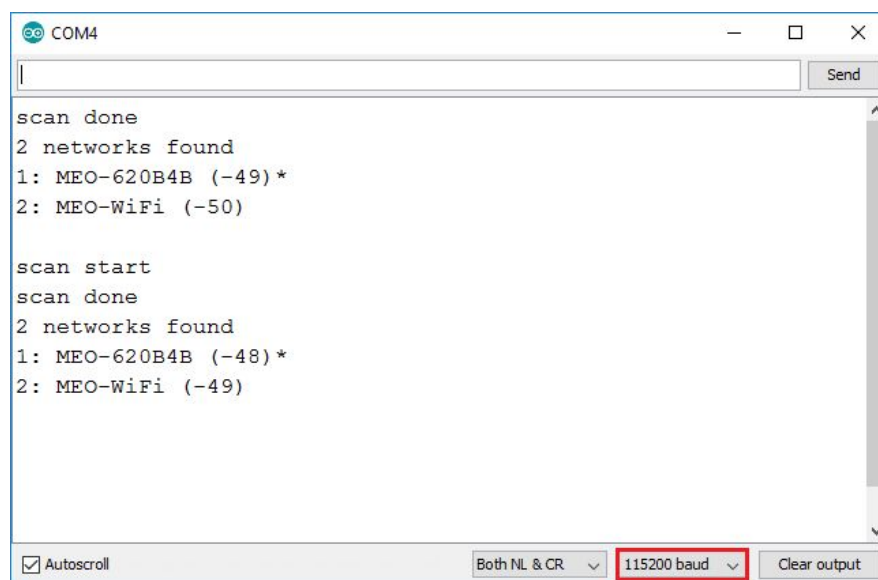
```
Done uploading.
Writing at 0x00040000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (e
Hash of data verified.

Leaving...
Hard resetting...
```

Figura 9: Done uploading do Arduino IDE

Agora, uma vez que a ESP32 esteja conectada ao computador, basta compilar o código e colocá-lo na ESP. Uma mensagem de carregamento sem fim pode aparecer nesse processo (na região preta e com texto laranja da IDE), caso isso aconteça é necessário pressionar o botão físico BOOT da ESP. Ao fim do carregamento, a mensagem ‘done uploading’ (carregado) irá aparecer na faixa verde.



```
COM4
scan done
2 networks found
1: MEO-620B4B (-49)*
2: MEO-WiFi (-50)

scan start
scan done
2 networks found
1: MEO-620B4B (-48)*
2: MEO-WiFi (-49)
```

Figura 10: Monitor serial do Arduino IDE

Para verificar o funcionamento do teste é necessário abrir o monitor serial da IDE. Em seguida, é necessário clicar no botão físico enable (EN) da ESP. Os resultados devem aparecer da seguinte forma, com a opção 115200 baud selecionada na taxa de atualização:

Após todo o processo de instalação do ESP32 na IDE, foi definida as bibliotecas utilizadas

```
#include <WiFi.h>
#include <PubSubClient.h>
```

Figura 11: Bibliotecas usados no ESP32

Sendo `#include <WiFi.h>` a biblioteca usada para fazer as conexões com o Wi-Fi e `#include <PubSubClient.h>` a biblioteca para fosse possível enviar e receber mensagens pelo MQTT.

Os pinos para conexão dos relés e do sensor e as variáveis para guardar valores

```
int pinoI = 32;
int pinoC1 = 33;
int pinoM = 25;
int pinoC2 = 26;

int status_I;
int status_C1;
int status_M;
int status_C2;

int porta_rele1 = 17; //Pino ligado ao Relé (ativado lamapada)
int acionamento; //Variavel para guardar valor do sensor
```

Figura 12: Definição dos pinos e variáveis

As portas 32, 33, 25 e 26 do ESP32 foram definidos, cada um para uma lâmpada como está nas quatro primeiras linhas. As variáveis `int status_I`; `int status_C1`; `int status_M`; `int status_C2` tem a função de armazenar o estado da lâmpada. Foi definido que na porta 17 está conectado o relé e a variável `int acionamento` tem a função de guardar o valor do sensor de presença.

Os tópicos de comunicação MQTT, cada uma das linhas acima, representa a definição de um tópico do MQTT, do qual será responsável por uma lâmpada.

```

#define TOPICO1      "letras/i"
#define TOPICO2      "letras/c1"
#define TOPICO3      "letras/m"
#define TOPICO4      "letras/c2"
#define TOPICO5      "comodo1/sensor1"

```

Figura 13: Conexão com os tópicos

Esse tópico é definido para especificar o Broker do MQTT.

```

#define ID_MQTT      "esp32_mqtt"

```

Figura 14: Definição do Broker

Foi feita a especificação dos dados necessários para conectar o Wi-Fi e o MQTT.

```

const char* SSID = "TP-Link_1B7A"; // SSID / nome da rede WI-FI que deseja se conectar
const char* PASSWORD = "71289168"; // Senha da rede WI-FI que deseja se conectar

const char* BROKER_MQTT = "192.168.0.100"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
int status_luz1;

```

Figura 15: Especificação do Wi-Fi e MQTT

```

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

```

Figura 16: Objeto espClient

```

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);

```

Figura 17: Funções usadas

```

void initWiFi(void)
{
    delay(10);
    Serial.println("-----Conexao WI-FI-----");
    Serial.print("Conectando-se na rede: ");
    Serial.println(SSID);
    Serial.println("Aguarde");

    reconnectWiFi();
}

```

Figura 18: Conexão Wi-Fi

Função para conectar com a rede do local

```

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)
}

```

Figura 19: Conexão MQTT

Função para indicar o Broker que deve se conectar e detecta quando uma mensagem chega a ele

```

void altera_lampada(String msg, char* topic){
    if(!strcmp(topic, TOPICO1)){

        if(msg.equals("0")) status_I=0;
        if(msg.equals("1")) status_I=1;
        if(msg.equals("2")) status_I=2;

        if(msg.equals("0")) Serial.println("I: desligado.");
        if(msg.equals("1")) Serial.println("I: ligado.");
        if(msg.equals("2")) Serial.println("I: no sensor.");
    }
    if(!strcmp(topic, TOPICO2)){
        status_C1 = (int)msg[0];

        if(msg.equals("0")) status_C1=0;
        if(msg.equals("1")) status_C1=1;
        if(msg.equals("2")) status_C1=2;

        if(msg.equals("0")) Serial.println("C1: desligado.");
        if(msg.equals("1")) Serial.println("C1: ligado.");
        if(msg.equals("2")) Serial.println("C1: no sensor.");
    }
    if(!strcmp(topic, TOPICO3)){
        status_M = (int)msg[0];

        if(msg.equals("0")) status_M=0;
        if(msg.equals("1")) status_M=1;
        if(msg.equals("2")) status_M=2;

        if(msg.equals("0")) Serial.println("M: desligado.");
        if(msg.equals("1")) Serial.println("M: ligado.");
        if(msg.equals("2")) Serial.println("M: no sensor.");
    }
    if(!strcmp(topic, TOPICO4)){
        status_C2 = (int)msg[0];

```

Figura 20: Define o estado da lâmpada

```

    }
    if(!strcmp(topic, TOPICO4)){
        status_C2 = (int)msg[0];

        if(msg.equals("0")) status_C2=0;
        if(msg.equals("1")) status_C2=1;
        if(msg.equals("2")) status_C2=2;

        if(msg.equals("0")) Serial.println("C2: desligado.");
        if(msg.equals("1")) Serial.println("C2: ligado.");
        if(msg.equals("2")) Serial.println("C2: no sensor.");
    }
}

```

Figura 21: Define o estado da lâmpada

Na figura 10 e 11 está uma estrutura de decisão feita com if para que a partir da mensagem recebida pelo MQTT, seja determinado o estado da lâmpada.

```

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

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

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

    altera_lampada(msg, topic);
}

```

Figura 22: Leitura da mensagem enviada pelo MQTT

Função que a string msg, o que coleta a mensagem que é enviada pelo MQTT e o for para ler a mensagem

```

void reconnectMQTT(void)
{
    while (!MQTT.connected())
    {
        Serial.print("* Tentando se conectar ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT))
        {
            MQTT.subscribe(TOPICO1);
            MQTT.subscribe(TOPICO2);
            MQTT.subscribe(TOPICO3);
            MQTT.subscribe(TOPICO4);

            Serial.println("Conectado com sucesso ao broker MQTT!");
        }
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}

```

Figura 23: Conexão MQTT

Função que faz a conexão com o Broker MQTT, dentro do while é feito a conexão com o Broker e com os tópicos, se a conexão não for realizada, a parte do else entrará em execução e após 2 minutos haverá uma nova tentativa.

```

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
}

```

Figura 24: Análise das conexões

Função que analisa a conexão com o Broker e com o Wi-Fi, se não houver ela será refeita


```

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("IP obtido: ");
    Serial.println(WiFi.localIP());
}

```

Figura 25: Refaz a conexão Wi-Fi se for preciso

```

void seta_I(int stat){
    status_I = stat;
}

void seta_C1(int stat){
    status_C1 = stat;
}

void seta_M(int stat){
    status_M = stat;
}

void seta_C2(int stat){
    status_C2 = stat;
}

```

Figura 26: Objetos

Essas funções faz de cada status da lâmpada um objeto.

```

void att_pinos(){
    if(status_I == 0) digitalWrite(pinoI, LOW);
    if(status_C1 == 0) digitalWrite(pinoC1, LOW);
    if(status_M == 0) digitalWrite(pinoM, LOW);
    if(status_C2 == 0) digitalWrite(pinoC2, LOW);

    if(status_I == 1) digitalWrite(pinoI, HIGH);
    if(status_C1 == 1) digitalWrite(pinoC1, HIGH);
    if(status_M == 1) digitalWrite(pinoM, HIGH);
    if(status_C2 == 1) digitalWrite(pinoC2, HIGH);

    if(acionamento == LOW){
        if(status_I == 2) digitalWrite(pinoI, LOW);
        if(status_C1 == 2) digitalWrite(pinoC1, LOW);
        if(status_M == 2) digitalWrite(pinoM, LOW);
        if(status_C2 == 2) digitalWrite(pinoC2, LOW);
    }

    if(acionamento == HIGH){
        if(status_I == 2) digitalWrite(pinoI, HIGH);
        if(status_C1 == 2) digitalWrite(pinoC1, HIGH);
        if(status_M == 2) digitalWrite(pinoM, HIGH);
        if(status_C2 == 2) digitalWrite(pinoC2, HIGH);
    }
}

```

Figura 27: Definição quando a lâmpada acende e apaga

Na série de tomadas de decisão pode ver que quando o status_I/C1/M/C2 quando igual à 0, a luz será desligada, quando igual à 1, a luz será ligada e quando a variável acionamento (que é a variável que armazena o estado captado do sensor) está igual a LOW e o status_I/C1/M/C2 igual a 2, a luz é apagada, sendo a partir do sensor de presença, e quando o status_I/C1/M/C2 igual a 2 mas o acionamento igual à HIGH as luzes serão acesas.


```

void lePIR() {
    acionamento = digitalRead(porta_rele1); //Le o valor do sensor PIR
    if (acionamento == LOW) //Sem movimento, mantem rele desligado
    {
        Serial.println("Parado.");
        MQTT.publish(TOPICO5, "0");
    }
    else //Caso seja detectado um movimento, aciona o rele
    {
        Serial.println("Movimento!");
        MQTT.publish(TOPICO5, "1");
    }
}

```

Figura 28: Leitura do sensor

Função que realiza a leitura do sensor de presença e de acordo com o que for detectado, ele acende ou desliga a lâmpada

```

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

    seta_I(1);
    seta_C1(1);
    seta_M(1);
    seta_C2(1);

    /* Inicializa a conexao wi-fi */
    initWiFi();

    /* Inicializa a conexao ao broker MQTT */
    initMQTT();

    pinMode(pinoI, OUTPUT);
    pinMode(pinoC1, OUTPUT);
    pinMode(pinoM, OUTPUT);
    pinMode(pinoC2, OUTPUT);

    pinMode(porta_rele1, INPUT); //Define pino sensor como entrada

    att_pinos();
}

```

Figura 29: Função Setup

Função que define os pinos de entrada e saída e a utilização de algumas funções

```
void loop()
{
    VerificaConexoesWiFIEMQTT();

    MQTT.loop();

    lePIR();
    att_pinos();
    delay(1000);
}
```

Figura 30: Função que manter as conexões

Função mantém a conexão Wi-Fi, MQTT e ler o sensor PIR a cada minuto

Servidor

O servidor é responsável pelo envio de mensagens para o microcontrolador, ditando então o seu funcionamento e recebendo também as informações de presença. Pode estar hospedado em uma Raspberry, fazendo com que o funcionamento seja somente para a rede interna, a não ser que sejam abertas portas externas no roteador.

O servidor é também responsável pela comunicação e geração da interface de controle, com a qual o usuário interage.

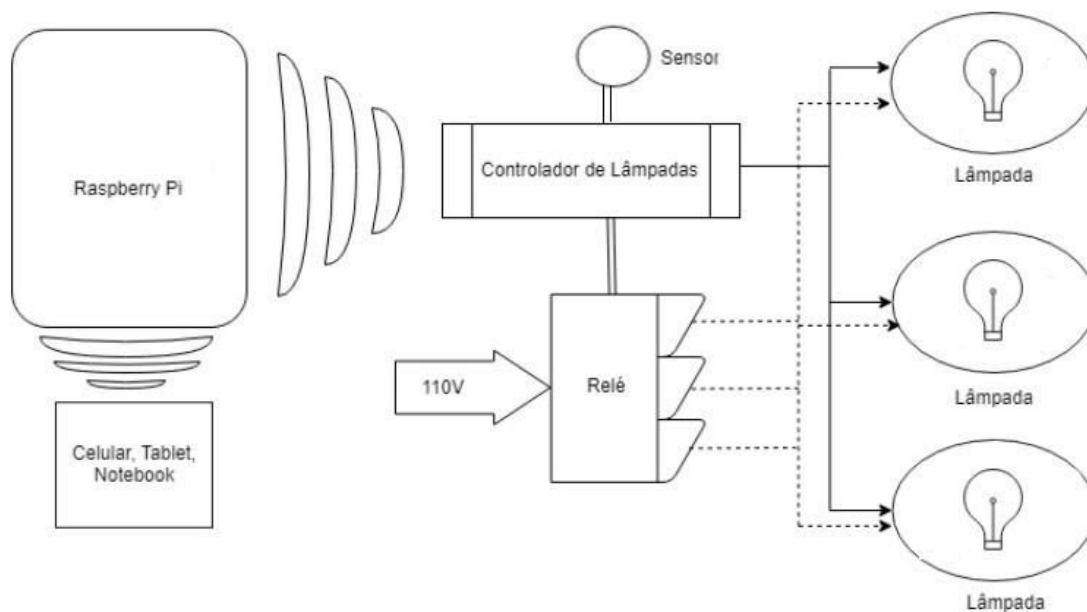


Figura 31: Funcionamento do projeto

Protocolo de comunicação

O protocolo de comunicação adotado foi o MQTT, o qual é um protocolo leve e rápido, ideal para soluções de IoT. Esse protocolo é baseado no conceito de Publish/Subscribe, que consiste em um dispositivo se cadastra (Subscribe) em um determinado Tópico e sempre que qualquer mensagem for publicada (Publish) nesse Tópico o dispositivo receberá a mensagem publicada. Esse protocolo necessita de um Broker que administrará os tópicos e garantirá o envio das mensagens.

Esse Broker estará no servidor, no nosso caso na raspberry. Existe um broker open-source disponibilizado pela Eclipse Foundation, chamado Mosquitto, o qual será utilizado no projeto.

A comunicação adotada no projeto funciona da seguinte forma, cada um dos cômodos constitui um tópico e cada uma das lâmpadas do cômodo constituem subtópicos, por exemplo “comodo1/luz1”, “comodo1/luz2”. A troca de mensagens é simples, no caso das lâmpadas, a mensagem “0” indica que a lâmpada deve ser apagada, “1” o acendimento da lâmpada e “2” o funcionamento da lâmpada em função do sensor de presença.

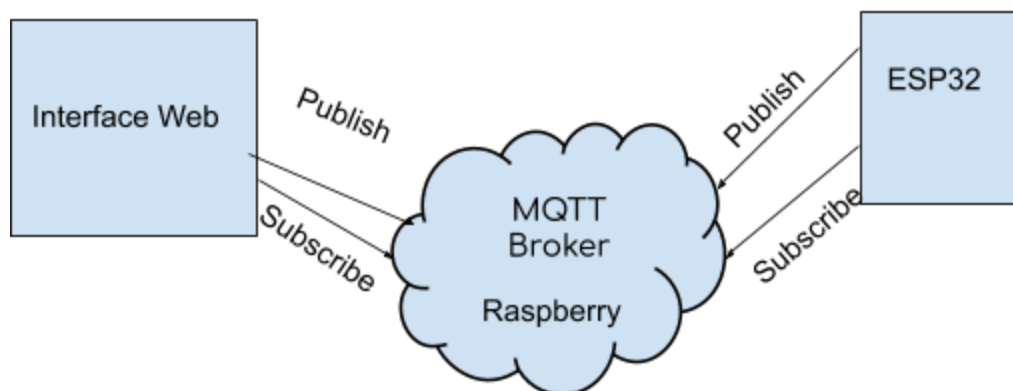


Figura 32: Funcionamento do MQTT

Plataformas

Arduino IDE suporta as linguagens C e C++ usando regras especiais de estruturação de código, foi utilizado para codificar o ESP 32 em linguagem C

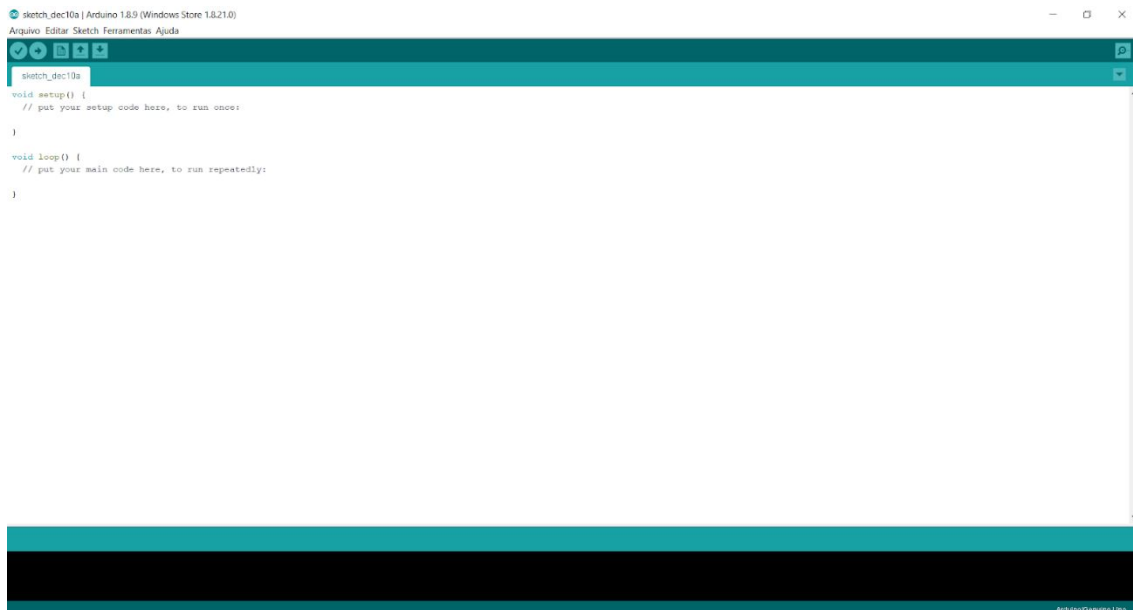


Figura 33: Arduino IDE

Node Red ferramenta de desenvolvimento com interface acessada pelo navegador, é baseada em fluxo para programação visual, com linguagem em JSON e JavaScript. Utilizada para programar a conexão MQTT entre a Raspberry e o ESP 32, os comandos e as resposta, fazer a interface e o bot do Telegram.

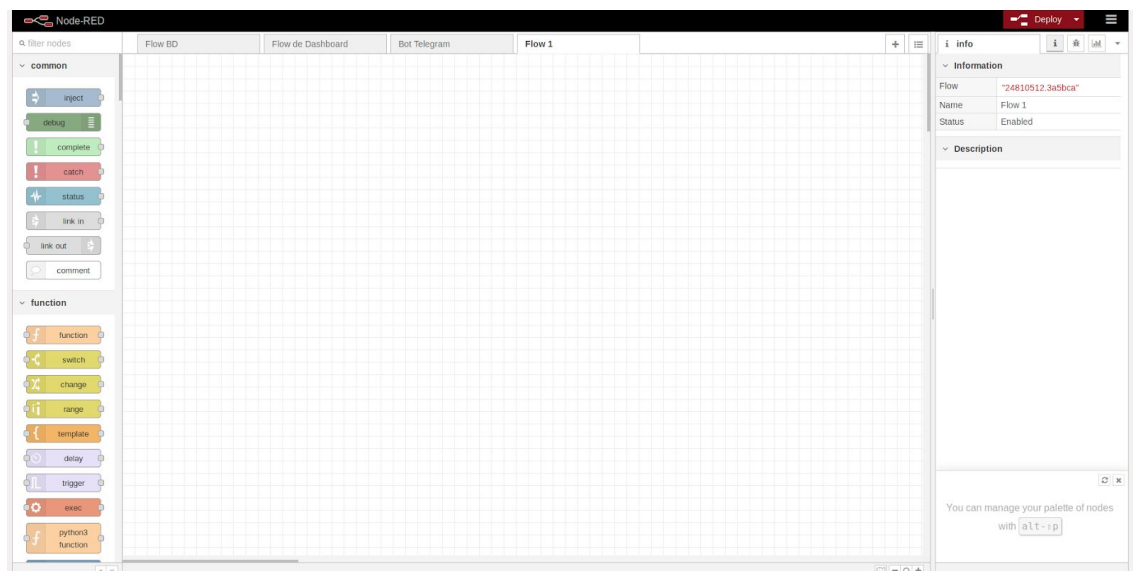


Figura 34: Plataforma Node Red

É necessária a instalação do broker Mosquitto e do node-red na raspberry, para isso os comandos executados no terminal devem ser:

```
sudo npm install -g --unsafe-perm node-red
sudo apt-get install mosquitto
```

Além disso, na plataforma do node-red é necessária a instalação de dois novos pacotes de nós, o node-red-dashboard e o node-red-contrib-chatbot. Para isso, basta clicar no menu, situado

no canto superior direito, em seguida em Manage Palates, e buscar ambos os nomes no campo de pesquisa e clicar no opção install.

Circuito

O circuito foi feito com o ESP 32 conectando o sensor PIR e os relés. Os pinos digitais de saída do ESP32 são utilizados para controlar os relés e os pinos digitais de entrada são usados para receber as leituras do sensor de presença PIR.

O circuito deve ser alimentado com tensão de 5V para o funcionamento do ESP e dos relés, por isso é também usada uma fonte para converter 110/220V para 5V.

Interfaces e relação com o usuário

As interfaces feitas neste projeto foram a dashboard que o Node Red disponibiliza, porém como trabalhamos com conexões do protocolo MQTT dentro de uma rede, para que a dashboard funcione, é preciso que o aparelho esteja conectado na mesma rede que o MQTT está trabalhando para envio e recebimento de mensagens.

Como isso pode ser um problema de comodidade para o usuário, foi pensado em uma segunda interface, que é um bot do Telegram (um aplicativo de comunicação que pode ser usado em aparelhos Android, IOs e em computadores), o qual consegue enviar e receber mensagens do protocolo e do estado das lâmpadas mesmo estando fora da rede de atuação no MQTT, portanto a usuário pode estar a quilômetros de distância da rede, mas se houver uma simples conexão com a internet, que pode ser móvel, ele pode saber o estado da lâmpada e controlá-la.

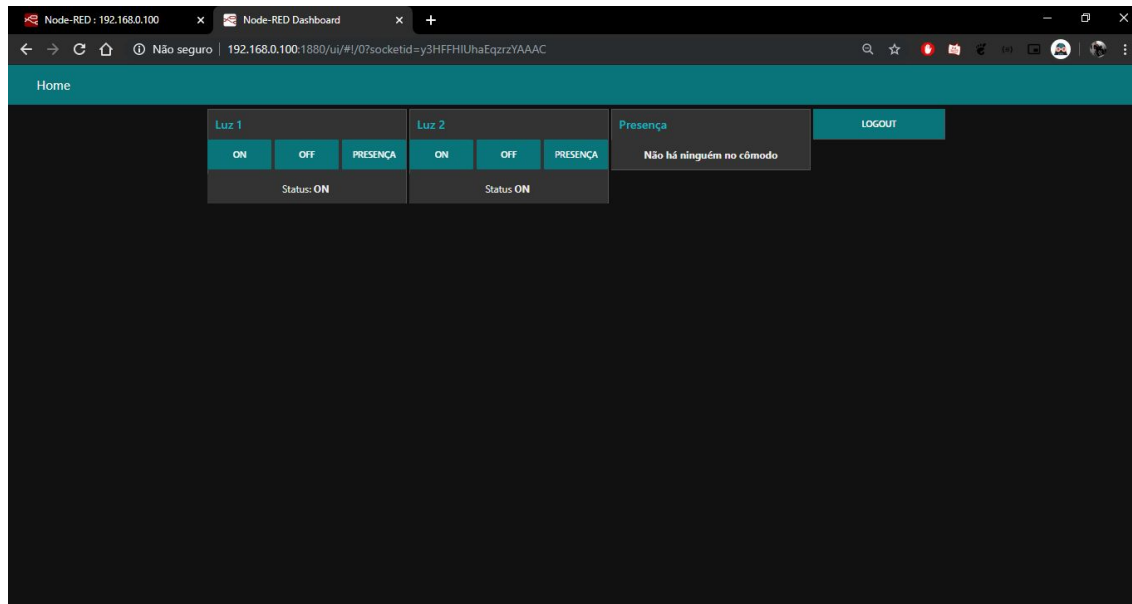


Figura 36: Interface da Dashboard

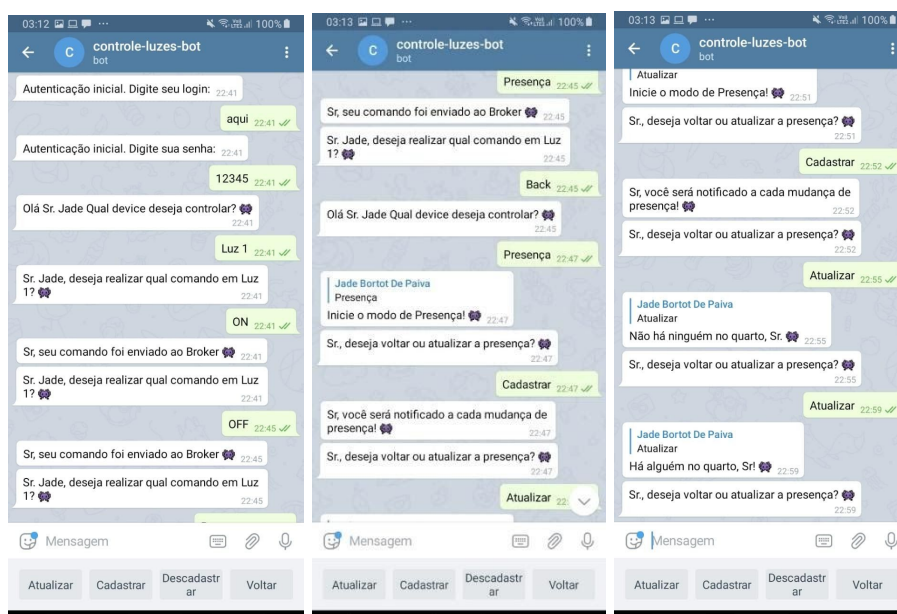


Figura 37: Interface do Bot do Telegram

Estas interfaces foram feitas utilizando o Node Red.

Este flow é referente ao Dashboard e a conexão MQTT realizada para o acionamento das luzes por botões e do sensor de presença.

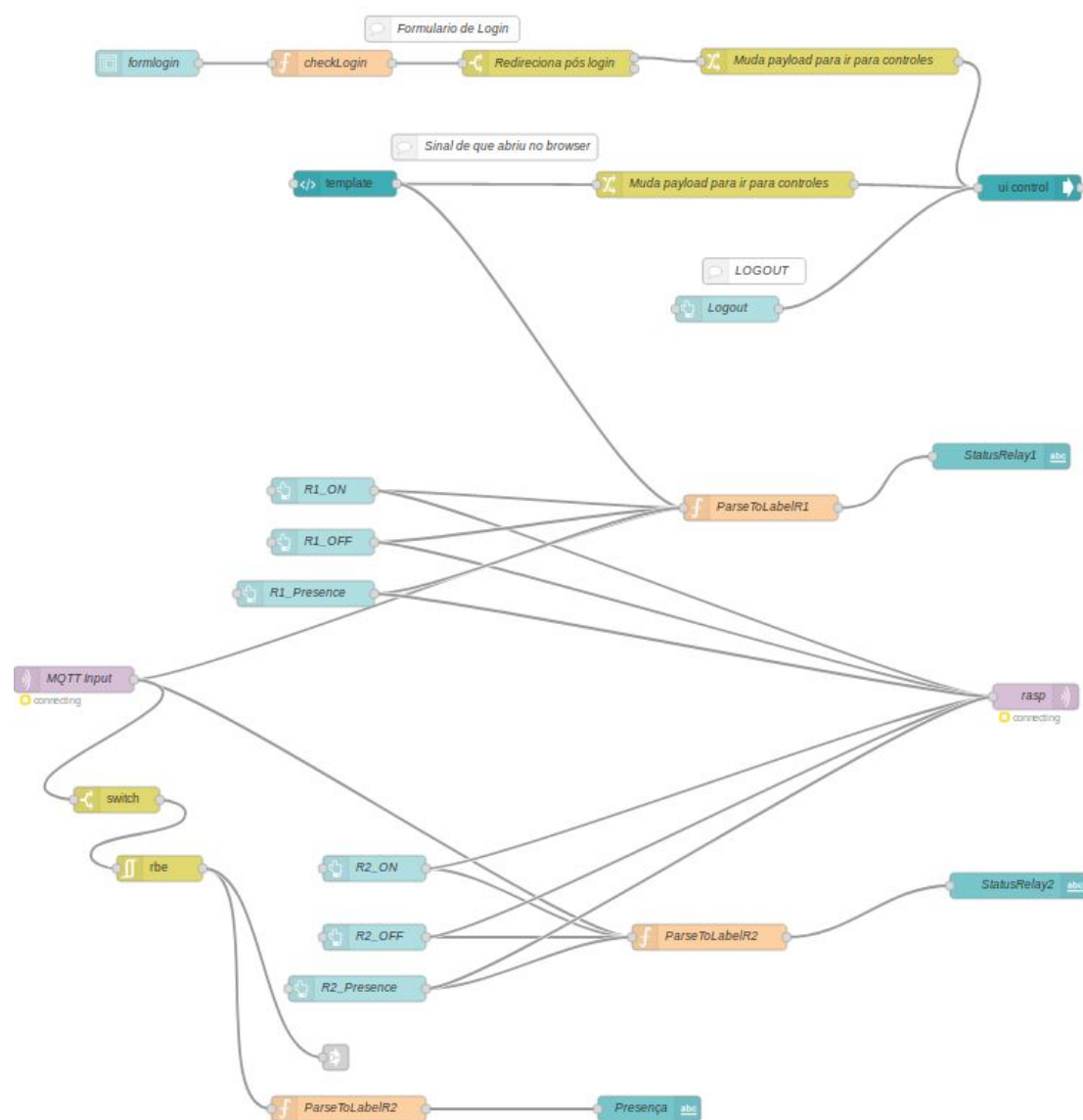


Figura 38: Flow feito no Node Red para a Dashboard

Sendo a primeira parte

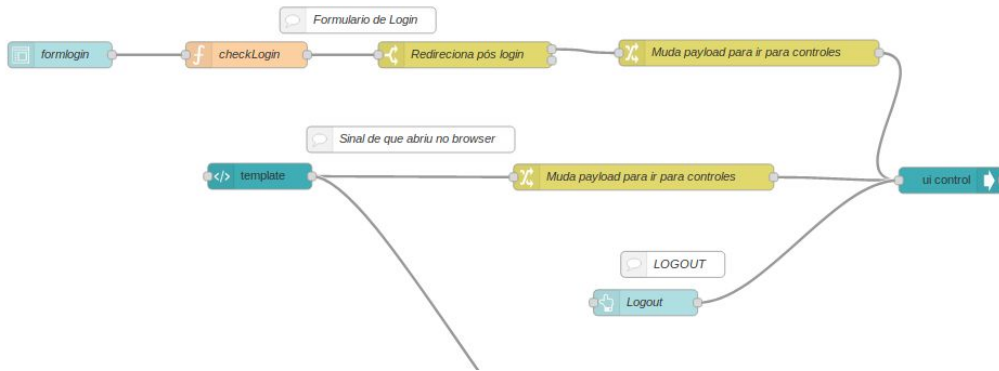


Figura 39: Parte do login do flow feito no Node Red

Feita para que o usuário realize o login para entrar na dashboard.

O formlogin é o formulário de login que deve ser preenchido pelo usuário para que seja possível acessar a interface; o checkLogin é uma função para chegar o login inserido; se estiver tudo correto, o usuário será redirecionado para tela de controle, isso é feito pelo Redireciona pós login e Muda payload para ir para controles. O balão template é o que constrói a configuração da interface, o logout é o botão presente na interface após login e o ui control é responsável pela alterações na interface gráfica, como a exibição ou ocultamento de informações na tela.

E a segunda parte

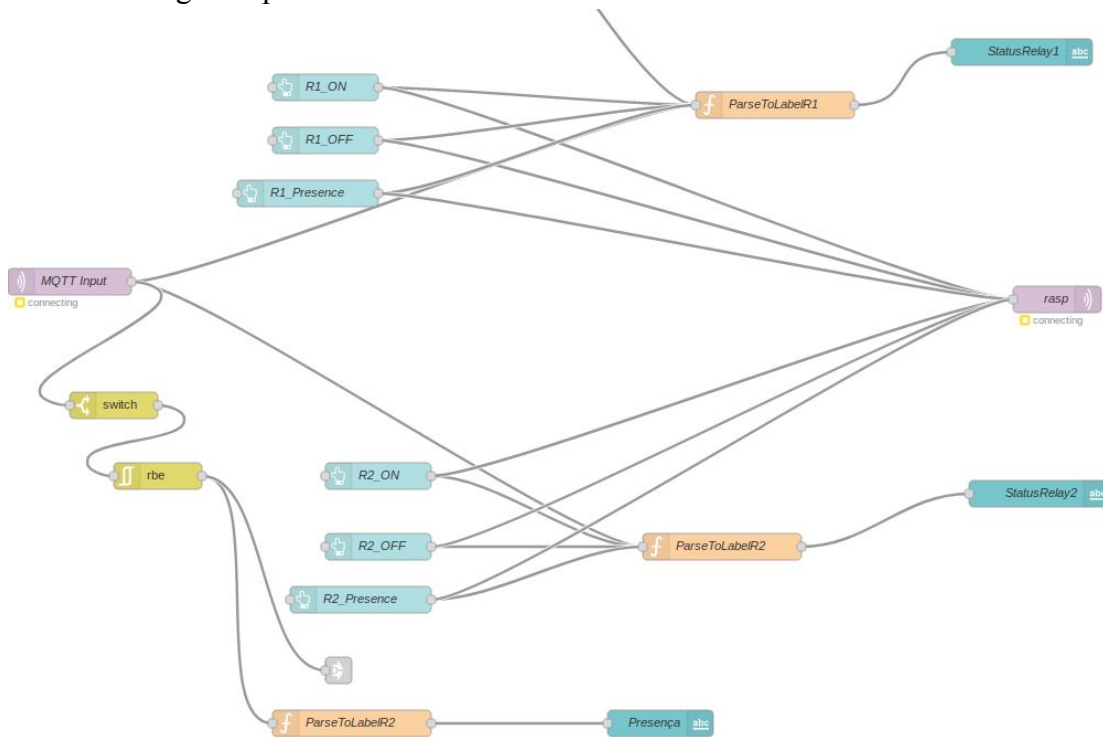


Figura 40: Parte do MQTT e botões do flow feito no Node Red

Da qual está realizando a conexão MQTT e configuração dos tópicos e insere os botões na dashboard, sendo eles, representados pelos balões R1_ON, R1_OFF, R1_Presence, R2_ON, R2_OFF, R2_Presence, dos quais defines do a lâmpada vai ser acesa, apagada ou se o sensor de presença será ativado, os balões ParseToLabelR1/R2 é uma função que analisa o estado da lâmpada e passa para para o balão StatusRelay1/2 para que esse estado seja visualizado na Dashboard.

Os balões switch e rbe são para transformar a mensagem recebida pelo balão MQTT Input em um objeto para que ele seja analisado pelo ParseToLabelR2 e seja exibido na interface pelo balão Presença. E o balão rasp é responsável por enviar mensagens através do MQTT para o servidor.

Já este é o Flow da interface do Bot do Telegram

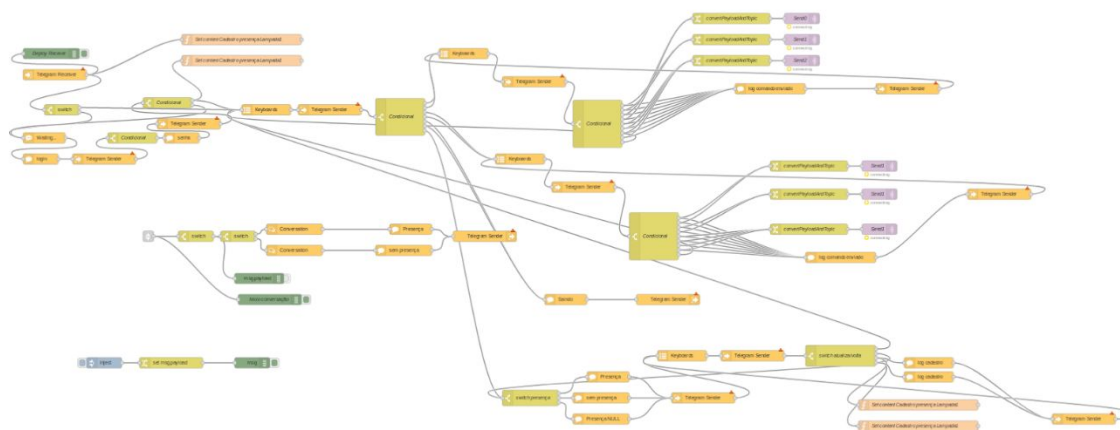


Figura 41: Flow feito no Node Red para o Bot do Telegram

Os códigos para o node-red e do ESP32 com o passo a passo detalhado no redme estão no GitHub, neste link:

Testes de funcionalidade

Na fase final foram feitos teste de funcionalidade com a solução pronta para uma baixa escala. Os testes foram feitos com duas luzes de led, o que não interfere no funcionamento com as lâmpadas, a simulação foi de apenas um cômodo da casa, para que o funcionamento da conexão MQTT fosse verificada, o qual está ocorrendo corretamente.

O sensor PIR apresenta dificuldades de estabilização, o que complica a detecção de presença mais rápida, o que complica a finalização da solução, porém foi encontrado uma configuração da qual o sensor se estabilizou e não foram percebidas demora na detecção de presença.

O cômodo simulado tem duas lâmpadas de funcionamento independentes e um sensor de presença, na interface, fizemos com que o usuário possa escolher se a lâmpada vai ser acesa pelo sensor PIR ou não, ele também pode escolher acender ou apagar a lâmpada por essa interface, a qual envia esses comando apenas se o aparelho, celular ou computador, estiver conectado na rede que o MQTT está atuando. Mas foi criado um bot no Telegram (um aplicativo de comunicação de celular) para que o usuário possa controlar as lâmpadas mesmo se ele estiver fora da rede de atuação do MQTT.

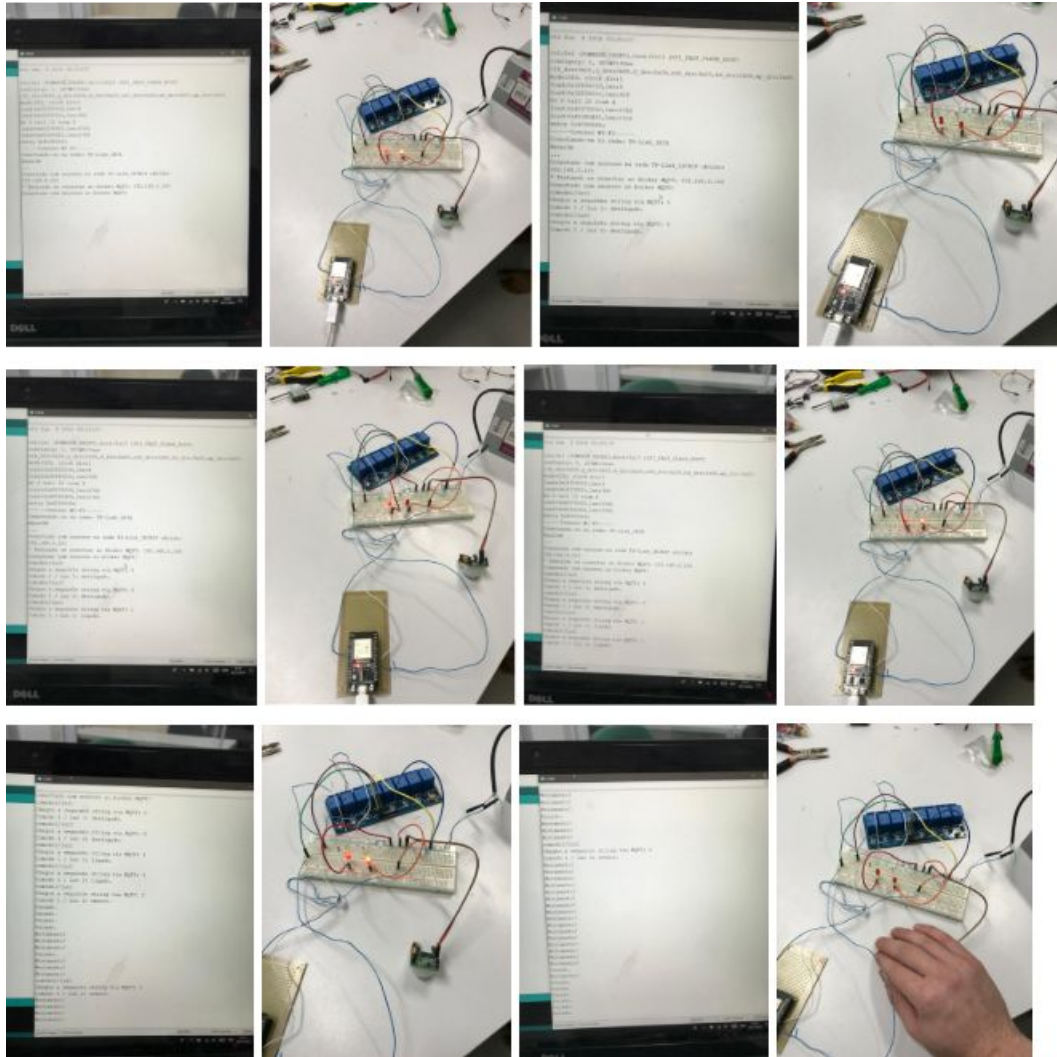


Figura 42: Sessão de teste finais

Sequência de testes realizados

- Estado inicial: todas as lâmpadas acesas;
- Apagando ambas as lâmpadas;
- Acendendo apenas uma lâmpada;
- Acendendo a outra lâmpada;
- Mantendo as lâmpadas acesas a partir do sensor de presença;
- Mantendo as lâmpadas apagadas a partir do sensor de presença;

O sistema conta com um sistema de login, o qual é necessário para a sua utilização. Além disso o status das lâmpadas é atualizado automaticamente, ao se pressionar os botões.

Conclusão

Ao término deste projeto, pode-se concluir que ocorreu tudo como o esperado e que este projeto é a base para a automação da iluminação de algum ambiente, existem pontos dos quais é

possível haver melhorias para isso, dos quais são facilmente implementados, como permitir que o usuário determine horários para o sensor de presença entrar em funcionamento, calcular o consumo de energia do ambiente, entre outros.