



INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e Gestão

Engenharia Informática

Ano Letivo 2024/2025

Internet das Coisas

Projecto final

Controlo Sanitário

Turno: D

Professor: Gustavo Silva Funchal

Aluno n.º a49618 – Óscar trindade Jose

Aluno n.º a49617 – Manelson jose Antóni

Índice

Introdução.....	2
Desenvolvimento	3
Páginas Node-Red:	3
InfluxDB (7 dias - Nível de CO2):	6
Node Red:.....	7
Código usado no ESP32:.....	10
Primeira parte do código:	10
Segunda parte do código(servo motores).....	12
Conclusão.....	18
Bibliografia	19

Introdução

O objetivo principal deste projeto é criar um ambiente mais seguro e confortável para os usuários do auditório. O sistema proposto inclui sensores para monitorar a presença de pessoas e a qualidade do ar, além de atuadores que ajustam automaticamente a ventilação e outros parâmetros ambientais com base nos dados coletados. A integração de serviços externos, como OpenWeather e IPMA, permite complementar as medições internas com informações ambientais externas, garantindo uma gestão mais completa e precisa.

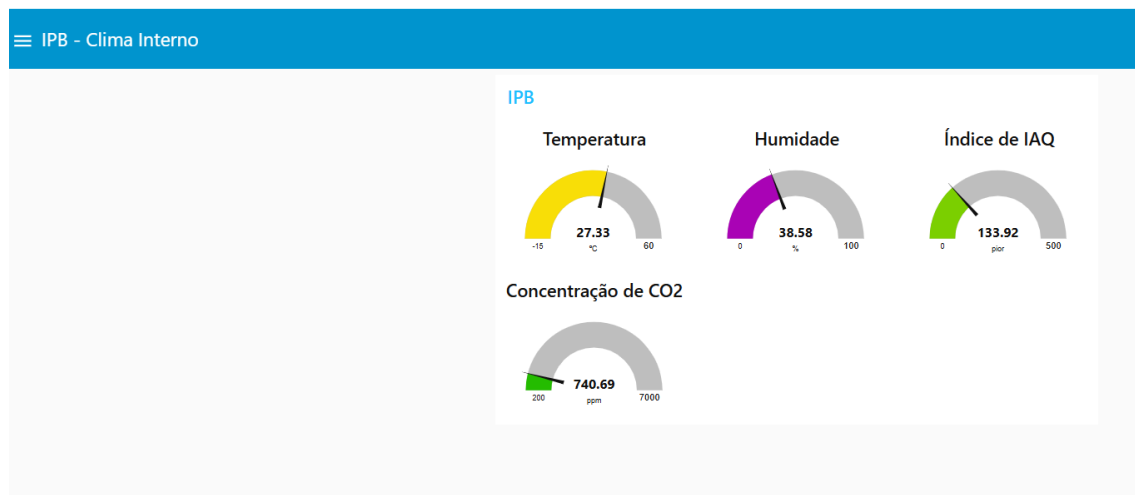
O uso de um microcontrolador ESP8266, juntamente com a plataforma Node-RED e um broker MQTT público, permite a criação de um sistema robusto e escalável. Um dashboard interativo oferece uma visualização em tempo real dos dados, possibilitando o acompanhamento histórico e a tomada de decisões informadas.

Desenvolvimento

Páginas Node-Red:

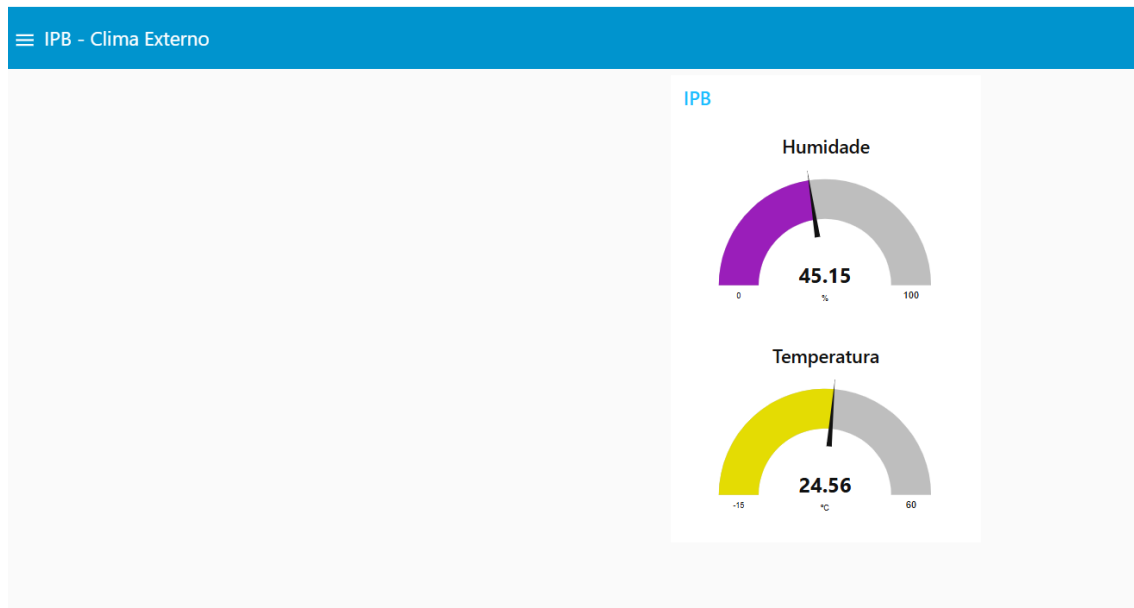


Na página Main é mostrada os dados do clima local direto do wokwi (também funcionadno fisicamente), mostrando os dados de humidade, luminosidade, temperatura e a lotação de pessoas pedidas no enunciado (mostrando uma mensagem se está ou não disponível lugares do auditório).

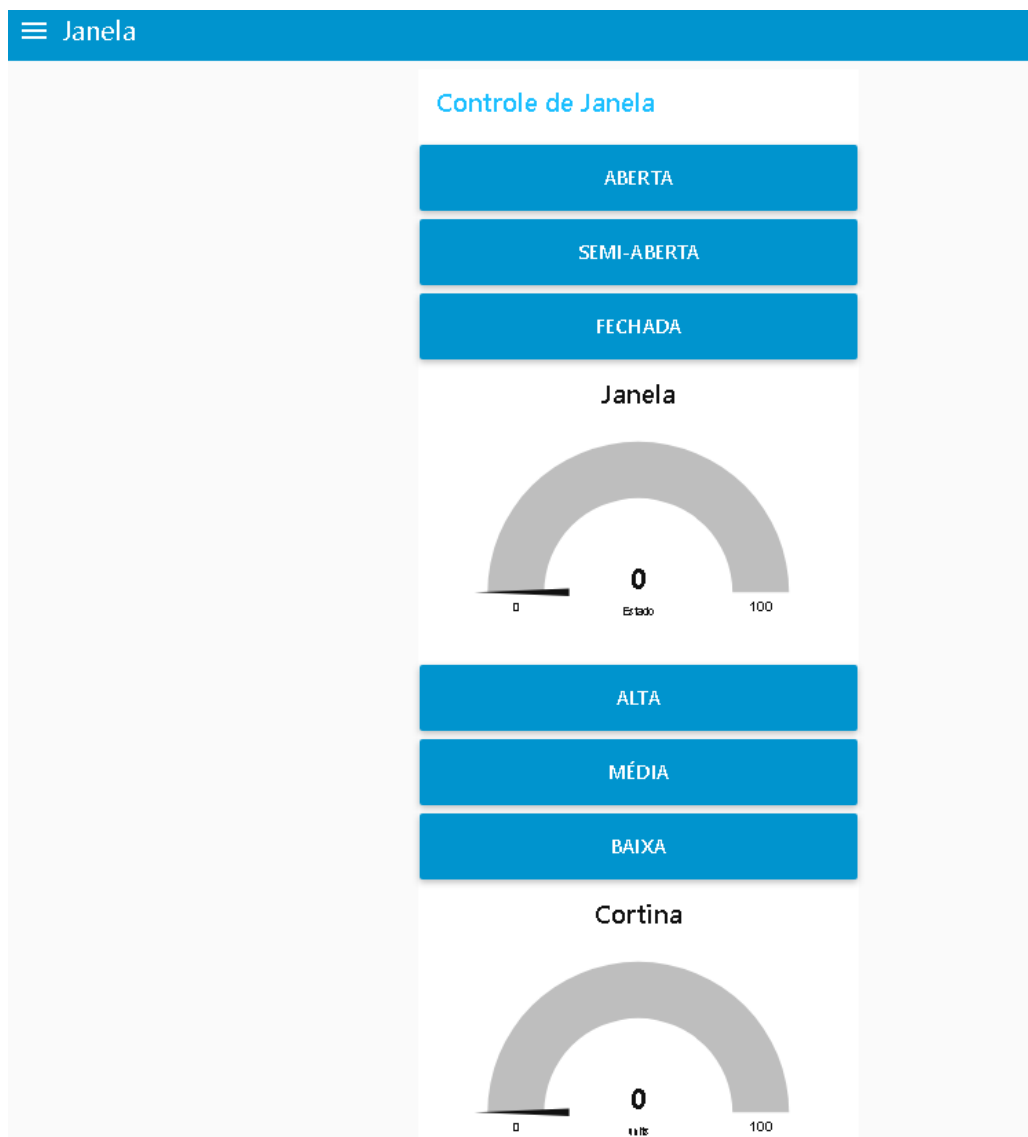


Nesta página são mostrados os dados de humidade, temperatura, IAQ e CO2 analisados no clima interno do IPB (Laboratório). Uma observação importante que fizemos nessa página foi na concestração de CO2, como podemos ver na imagem, os valores variam entre 200 e 7000, e a explicação é simples, com a ajuda de uma pesquisa que fizemos, o nosso planeta no obteve o mínimo de CO2 de todos os tempos com o valor de 200-225 na idade da pedra, no dias que vivemos hoje precisariamos de que não houvesse nenhum tipo de poluição atmosférica para chegarmos a esse valor, sendo que isso nas cidades hoje em dia não seja uma tarefa fácil de se realizar.

O valor 7000 está como o máximo porque os valores que podem ser prejudiciais para o ser humano é de 6000-7000.



Nesta página é mostrada os dados da temperatura e humidade externa do IPB.

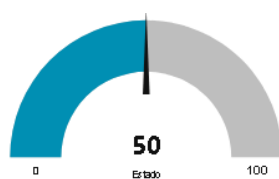


A imagem acima é um controlo de janela, mostrando num gauge as posições da janela e da cortina:

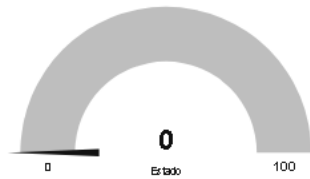
Fechado/Baixa (Roda 180° no servo-motor):



Semi-Aberta/Média (Roda 90° no servo-motor):



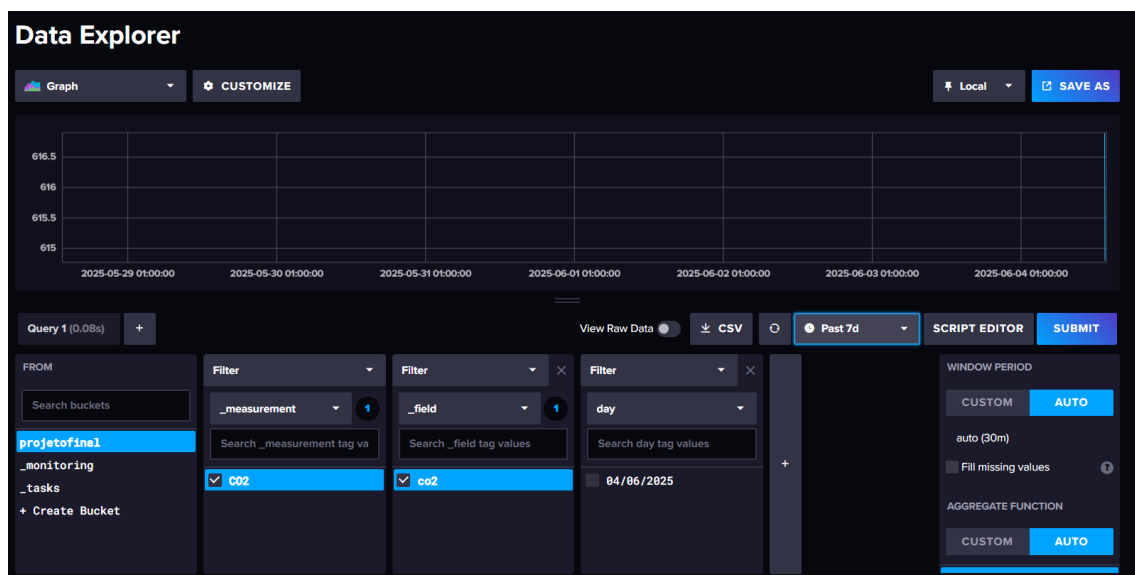
Aberta/Alta (Mantém a posição em 0°):



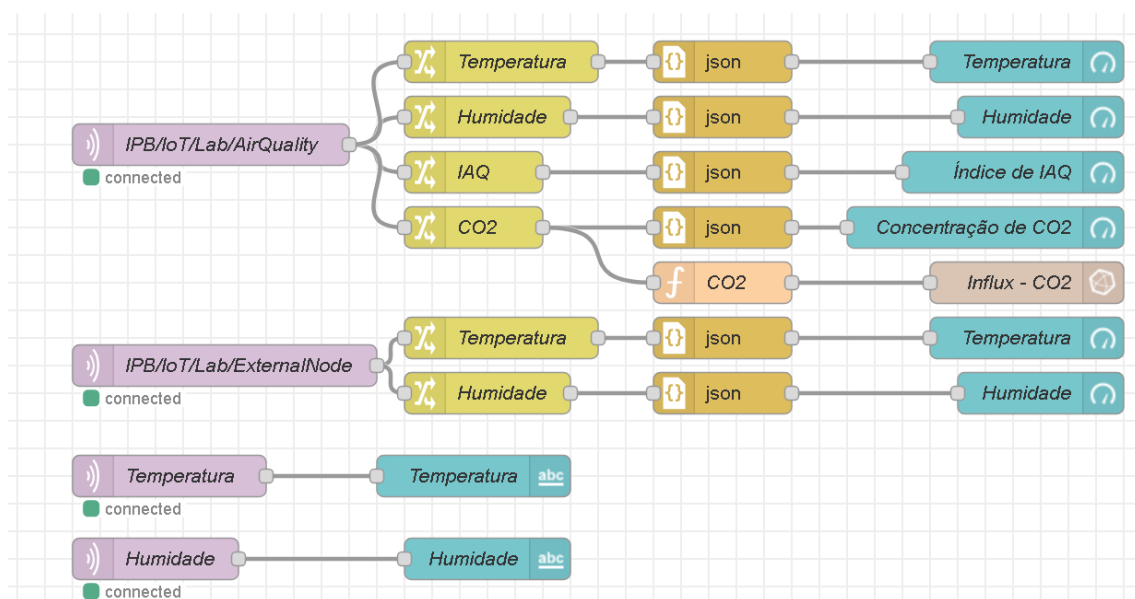
InfluxDB (7 dias- Nível de CO2):

Na imagem a seguir será mostrada no nível de CO2 presente no clima interno do IPB nos últimos 7 dias, O InfluxDB teve a capacidade de importar os dados do nível de CO2 do AirQuality e apresentar em line chart os resultados:

Para que os resultados fossem constantes, o programa teria que estar ligado num periodo de 7 dias para mostrar os dados da última semana.



Node Red:



Para mostrar os dados do clima interno e externo do ipb utilizamos os tópicos do enunciado e pegamos a mensagem de cada atributo que verificamos no debug por exemplo:

Name

Temperatura

Rules

Set

▼ msg. payload

to the value

▼ msg. payload.temp

Na parte do Influx pegamos o atributo “co2_eqv” e fizemos uma função para que fosse mostrada na plataforma.

Função CO2:

```
var co2Data = msg.payload;
var currentDate = new Date();
var day = currentDate.getDate().toString().padStart(2, '0');
var month = (currentDate.getMonth() + 1).toString().padStart(2, '0'); // Months are zero-based
var year = currentDate.getFullYear();
var currentDay = `${day}/${month}/${year}`;

var msg = {
  payload: [
    {
      measurement: "CO2",
      tags: {
        day: currentDay
      },
      fields: {
        co2: co2Data
      }
    }
  ]
}


return msg;
```

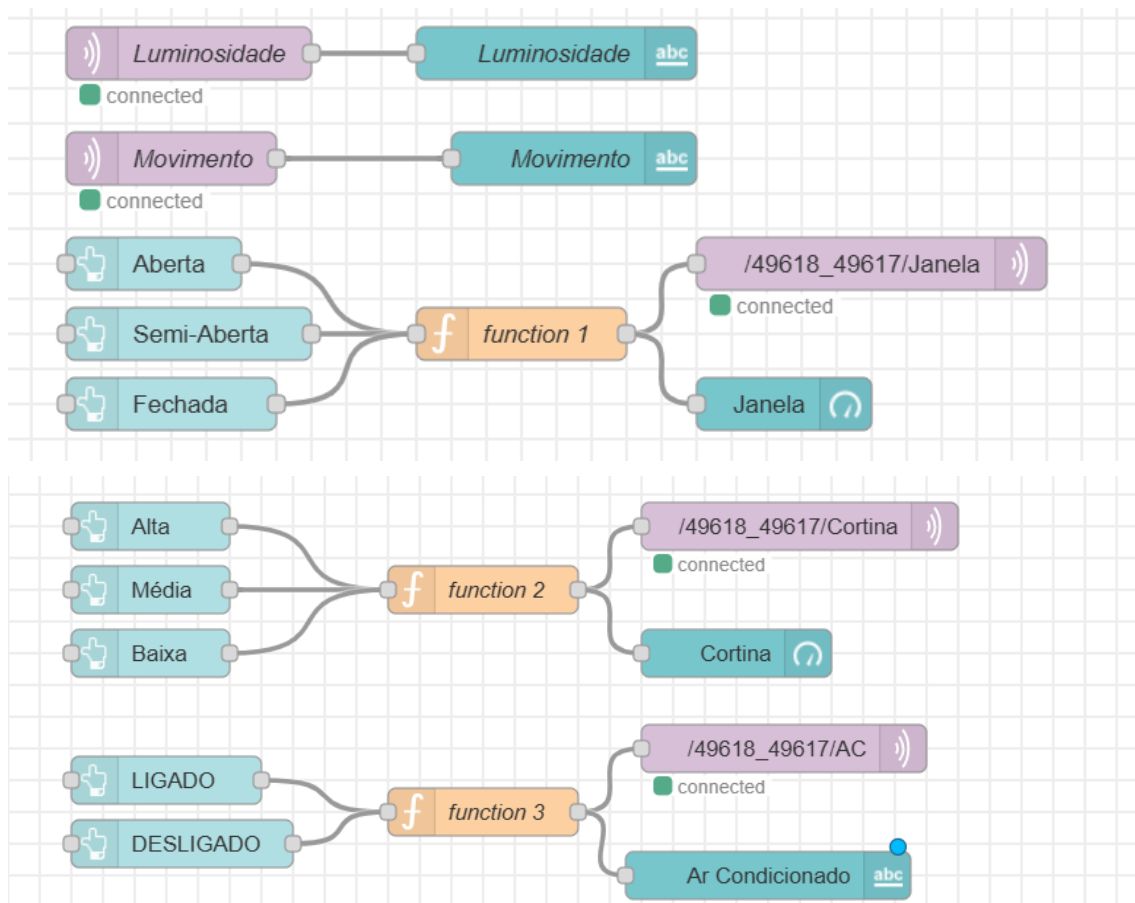
Influx:

 Name	<input type="text" value="Influx - CO2"/>	
 Server	<input type="text" value="[v2.0] http://localhost:8086"/>	<input type="button" value="v"/>
 Organization	<input type="text" value="ipb"/>	
 Bucket	<input type="text" value="projetofinal"/>	

Nos tópicos input mqtt dos dados do hardware criamos um tópico para serem usados mais tarde no wokwi, assim os clientes podem consultar os dados do clima local e da lotação do auditório.

Por exemplo o tópico “Temperatura”:

 Server	<input type="text" value="broker.hivemq.com:1883"/>	<input type="button" value="v"/>	<input type="button" value="p"/>	<input type="button" value="+"/>
Action	<input type="text" value="Subscribe to single topic"/>			
 Topic	<input type="text" value="/49618_49617/temperature"/>			



Para os botões dos controlos, apenas precisamos ligar os botões de cada funcionalidade em funções que apenas retornam mensagem "return msg;", e em cada botão foi definido um valor numérico para depois no wokwi conseguir identificar o número que o botão representa para mandar o sinal para os servo-motores, por exemplo, o botão "Aberta":

✉ When clicked, send:

Payload

▼ 0₉ 0

Topic

▼ msg. topic

Código usado no ESP32:

Primeira parte do código:

Nesta primeira parte do código, podemos observar que são incluídas as bibliotecas necessárias e também é definido os servos motores (Servo

myservoJanela Servo myservoCortina Servo myservoAC), configuramos também a rede WI-FI e o servidor MQTT , é inicializado também o sensor DHT e definido o seu pino, por fim são declaradas variáveis para armazenar a temperatura, humidade, lotação.

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

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

```
#include <ESP32Servo.h>
```

```
#include "DHTesp.h"
```

```
Servo myservoJanela; // Servo para janela
```

```
Servo myservoCortina; // Servo para cortina
```

```
Servo myservoAC; // Servo para AC
```

```
const char* ssid = "Wokwi-GUEST";
```

```
const char* password = "";
```

```
const char* mqtt_server = "broker.hivemq.com"; // Servidor MQTT
```

```
WiFiClient espClient;
```

```
PubSubClient client(espClient);
```

```
DHTesp dhtSensor;
```

```
const int DHT_PIN = 15;
```

```
#define LIGHT_SENSOR_PIN 36 // ESP32 pin GPIO36 (ADC0)
```

```
#define BUZZER_PIN 27 // Pino do buzzer
```

```
#define PIR_PIN 35 // Pino do sensor PIR
```

```
float temp = 0;
```

```
float hum = 0;
```

```
long lastMsg = 0;
```

```
char msg[50];
```

```
int value = 0;
```

```
int lotacao = 0; // Variável de lotação
```

Segunda parte do código(servo motores)

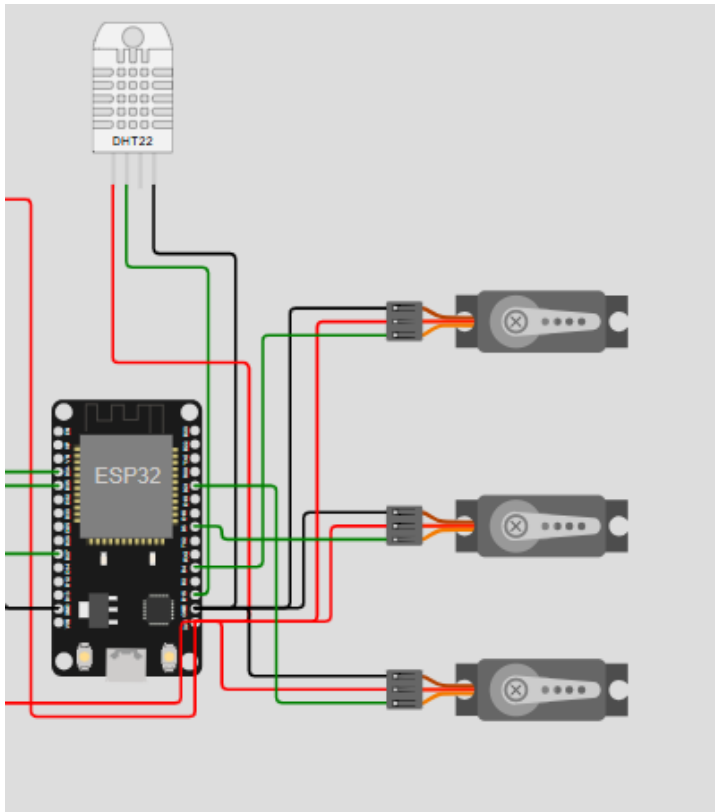
Na segunda parte do código pertence ao servo motores através do node-red podemos controlar a janela e a cortina nas posições Abertas Semi-abertas e Fechadas, também podemos controlar o AC nas posições Abertas e Fechadas são definidos os pinos pertencentes a cada um dos servo:

```
myservoJanela.attach(5); // Servo para janela no pino 5
```

```
myservoCortina.attach(4); // Servo para cortina no pino 4
```

```
myservoAC.attach(21); // Servo para AC no pino 21
```

É conectado também o ESP32 à rede wifi, cada um dos servo motores está ligado ao respetivo tópico no node-red através(("49618_49617/Janela"))



Node-red botões:

```
void setup() {  
  Serial.begin(115200);  
  
  myservoJanela.attach(5); // Servo para janela no pino 5  
  myservoCortina.attach(4); // Servo para cortina no pino 4  
  myservoAC.attach(21); // Servo para AC no pino 21  
  
  dhtSensor.setup(DHT_PIN, DHTesp::DHT22);  
  
  pinMode(BUZZER_PIN, OUTPUT);  
  pinMode(PIR_PIN, INPUT);
```

```
    setup_wifi();

    client.setServer(mqtt_server, 1883);

    client.setCallback(callback);
}

void setup_wifi() {

    delay(10);

    Serial.println();

    Serial.print("Connecting to ");

    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {

        delay(500);

        Serial.print(".");

    }

    Serial.println("");

    Serial.println("WiFi connected");

    Serial.println("IP address: ");

    Serial.println(WiFi.localIP());

}

void callback(char* topic, byte* payload, unsigned int length) {

    String string;
```

```
Serial.print("Message arrived [");
```

```
Serial.print(topic);
```

```
Serial.print("] ");
```

```
for (int i = 0; i < length; i++) {
```

```
    string += ((char)payload[i]);
```

```
}
```

```
Serial.println(string);
```

```
if (strcmp(topic, "/49618_49617/Janela") == 0) { // Verifica o tópico da janela
```

```
    Serial.print(" ");
```

```
    int status = string.toInt();
```

```
    int pos = map(status, 1, 100, 0, 180);
```

```
    Serial.println(pos);
```

```
    myservoJanela.write(pos);
```

```
    delay(15);
```

```
}
```

```
if (strcmp(topic, "/49618_49617/Cortina") == 0) { // Verifica o tópico da cortina
```

```
    Serial.print(" ");
```

```
    int status = string.toInt();
```

```
    int pos = map(status, 1, 100, 0, 180);
```

```
    Serial.println(pos);
```



```
myservoCortina.write(pos);  
  
delay(15);  
  
}
```

```
if (strcmp(topic, "/49618_49617/AC") == 0) { // Verifica o tópico do AC
```

```
    Serial.print(" ");
```

```
    int status = string.toInt();
```

```
    int pos = map(status, 1, 100, 0, 180); // mapeamento
```

```
    Serial.println(pos);
```

```
    myservoAC.write(pos);
```

```
    delay(15);
```

```
    // Adicionando condição para retornar "ligado" ou  
    "desligado" baseado no valor de status
```

```
    if (status == 0) {
```

```
        Serial.println("ligado");
```

```
    } else if (status == 100) {
```

```
        Serial.println("desligado");
```

```
    }
```

```
}
```

```
}
```

```
void reconnect() {
```

```
    while (!client.connected()) {
```

```
        Serial.print("Attempting MQTT connection...");
```

```
        if (client.connect("ESPClient")) {
```

```
            Serial.println("connected");
```

```
            client.subscribe("/49618_49617/Janela");
```

```
            client.subscribe("/49618_49617/Cortina");
```

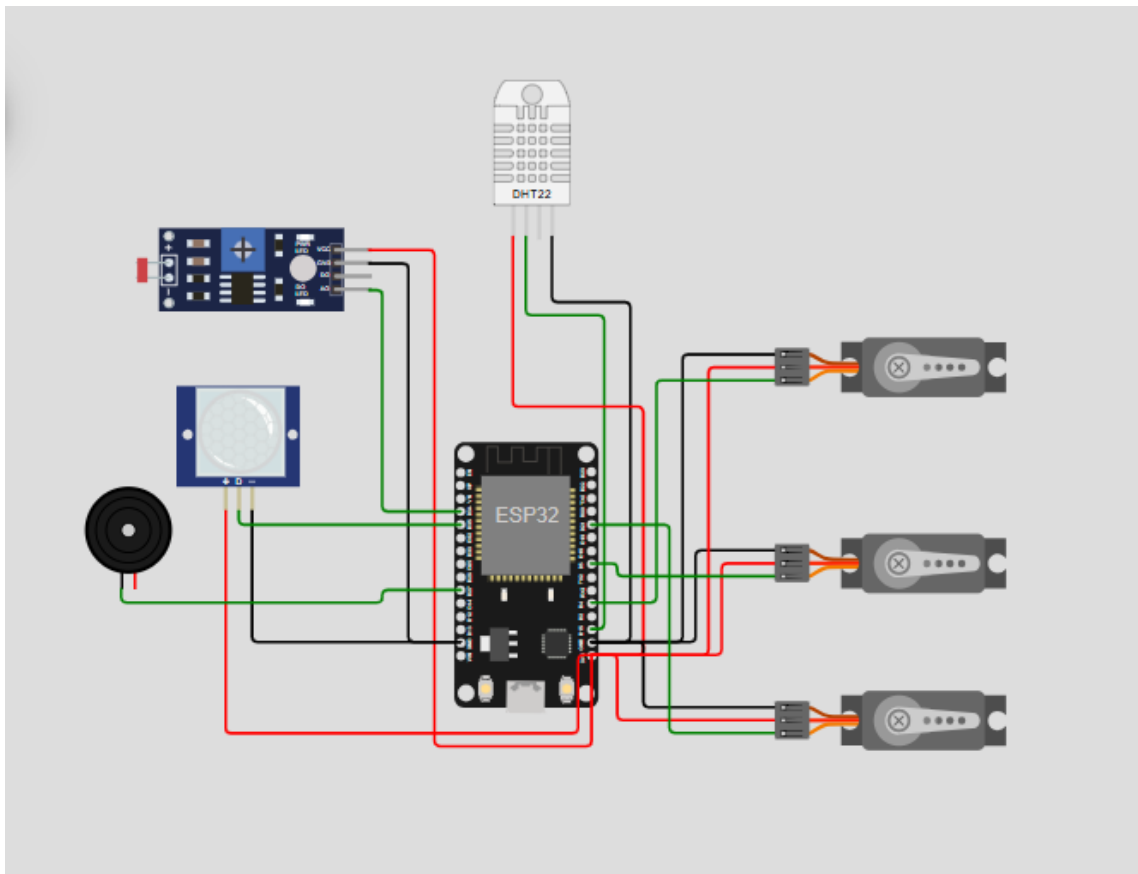
```
            client.subscribe("/49618_49617/AC");
```

```

    } else {
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" try again in 5 seconds");
        delay(5000);
    }
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
}

```



Conclusão

O projeto cumpriu com sucesso o objetivo de criar um ambiente mais seguro e confortável para os utilizadores do auditório. A implementação de sensores para monitoramento da presença de pessoas e da qualidade do ar, juntamente com atuadores que ajustam automaticamente a ventilação e outros parâmetros ambientais, resultou em um sistema eficiente e responsivo. A integração com serviços externos como OpenWeather e IPMA proporcionou uma gestão ambiental mais completa e precisa, complementando as medições internas com informações relevantes sobre as condições climáticas externas.

A escolha do microcontrolador ESP8266, aliado à plataforma Node-RED e a um broker MQTT público, demonstrou ser uma solução robusta e escalável. A utilização de um dashboard interativo permitiu uma visualização em tempo real dos dados coletados, facilitando o acompanhamento histórico e a tomada de decisões informadas pelos gestores do auditório.

Durante o desenvolvimento, foi possível observar a precisão e confiabilidade dos sensores, especialmente no monitoramento dos níveis de CO₂, cuja variação foi analisada em detalhes. A implementação dos servomotores para controle das janelas, cortinas e ar-condicionado funcionou conforme o esperado, permitindo ajustes automáticos com base nos dados recebidos.

Bibliografia

<https://institutolivres.org.br/o-medidor-tds-nao-nos-deixa-mentir/>

<https://zenzorcontrol.pt/pt/nivel-de-co2-como-detetar-quais-os-efeitos>

link de acesso GitHub

https://github.com/oscarchinene/Projeto_Final_IoT.git