

Trabalho Prático de Internet das Coisas e Ciências dos Dados

Prof. João Francisco Soares

Prof. Ricardo Gama

*“Sistema inteligente de deteção e alerta de fumo
baseado em sensores de dióxido de carbono”*

Trabalho realizado:

João Ferreira PV26017

José Pereira PV26019

Miguel Rodrigues PV26022

Índice

Introdução:	5
Apresentação do problema:	5
Objetivo do projeto:	5
Metodologia	6
Descrição do sistema:	6
Hardware:	6
Software:	6
Configuração dos sensores:	7
1. Montagem do Circuito:	7
2. Programação do ESP32:	7
3. Calibração do Sensor:	7
4. Teste do Sistema:	7
Transmissão de dados	7
1. Leitura de Dados	7
2. Processamento de Dados	7
3. Transmissão de Dados	7
4. Receção de Dados	7
5. Armazenamento de Dados:	7
Tentativas Práticas	8
Elaboração do Projeto	15
Montagem de uma maquete	24
Melhorias e direções futuras	25
Conclusão	26

Índice de Imagens

Figura 1 - Arduino com MQ135.....	8
Figura 2 - Código Arduino IDE	8
Figura 3- Código LoRa.....	9
Figura 4 - Código BLE (Bluetooth Low Energy)	10
Figura 5 - Leitura de Dados na Aplicação nRF	11
Figura 6 - ESP32 Rx<>Tx por BLE	11
Figura 7 - Código Sevidor MQTT	12
Figura 8 - Código Cliente MQTT	13
Figura 9 - Material utilizado	15
Figura 10 - Esquema dos 3 dispositivos elaborados.....	15
Figura 11 - As 3 salas de teste dos sensores	15
Figura 12 – Inclusão das Bibliotecas.....	16
Figura 13 - Definição de Constantes.....	16
Figura 14 - Definição de Variáveis	16
Figura 15 - Função disableBrownoutDetector	17
Figura 16 - Função Setup.....	17
Figura 17 - Função Loop	18
Figura 18 - Output obtido no Sensor1	19
Figura 19 - Dados Projeto Firebase	20
Figura 20 - Dados em RealTime Database.....	20
Figura 21 - Login e iniciação no Firebase Console.....	20
Figura 22 - Código Python a correr no Notebook	21
Figura 23 - Exportação dos dados para um ficheiro CSV.....	21
Figura 24 - Leitura do ficheiro CSV no Notebook	22
Figura 25 - Criação de um Gráfico Dados/Tempo para cada sensor	22
Figura 26 - Ficheiro CSV com dados exportados	23
Figura 27 - Maquete do Projeto	24

CONCEITOS

Arduino: é uma plataforma de eletrônica de código aberto baseada em hardware e software fáceis de usar. As placas Arduino são capazes de ler entradas e transformá-las em saída.

ESP32: é uma placa de desenvolvimento que possui todas as características necessárias para criar seus projetos. É superior ao Arduino UNO e ao ESP82663.

TTGO: O TTGO ESP32 é uma placa de desenvolvimento que integra um display TFT colorido. Esta placa é baseada no chip ESP32, que é um microcontrolador de 32 bits com capacidade de Wi-Fi e Bluetooth.

MQ135: é um sensor de gás disponível como um módulo pré-cabado ou apenas como a parte do sensor de gás.

MQTT: é um protocolo de mensagens baseado em padrões, usado para comunicação máquina a máquina.

LoRa: é uma técnica de modulação sem fio derivada da tecnologia Chirp Spread Spectrum (CSS). A transmissão modulada por LoRa é robusta contra perturbações e pode ser recebida a grandes distâncias.

BLE (Bluetooth Low Energy): é uma tecnologia sem fio de área pessoal projetada e comercializada pelo Bluetooth Special Interest Group (Bluetooth SIG) voltada para aplicações inovadoras nas indústrias de saúde, fitness, balizas, segurança e entretenimento doméstico.

Firestore: é um conjunto de serviços de computação em nuvem e plataformas de desenvolvimento de aplicativos fornecidos pelo Google. Ele hospeda bancos de dados, serviços, autenticação e integração para uma variedade de aplicativos.

Realtime Database: refere-se a um sistema de banco de dados que usa tecnologias de streaming para lidar com cargas de trabalho cujo estado está constantemente mudando. Isso difere dos bancos de dados tradicionais que contêm dados persistentes, em grande parte não afetados pelo tempo.

Jupyter Notebook: é uma aplicação web de código aberto que permite criar e compartilhar documentos que contêm código ao vivo, equações, visualizações e texto.

Python: é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi criada por Guido van Rossum em 1991. É muito utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018.

Introdução:

Apresentação do problema:

A detecção de fumo é uma questão crítica para a segurança e a saúde em muitos ambientes, especialmente em espaços fechados como residências, escritórios e fábricas. A exposição ao fumo pode levar a uma série de problemas de saúde, desde irritações nos olhos e no sistema respiratório até condições mais graves como doenças cardíacas e pulmonares. Em casos extremos, a inalação de fumo pode ser fatal.

Além disso, a detecção precoce de fumo é um componente essencial na prevenção de incêndios. Um sistema de detecção de fumo eficaz pode alertar os ocupantes de um edifício sobre um possível incêndio, permitindo que eles evacuem com segurança e alertem as autoridades para controlar o incêndio.

No entanto, muitos dos sistemas de detecção de fumo existentes são caros, difíceis de instalar ou requerem manutenção regular. Além disso, alguns desses sistemas podem não ser sensíveis o suficiente para detectar níveis baixos de fumo, ou podem ser propensos a falsos alarmes.

Portanto, existe uma necessidade de desenvolver um sistema de detecção de fumo que seja acessível, fácil de instalar e manter, e que forneça detecção precisa e confiável de fumo. Este é o problema que este projeto visa resolver.

Objetivo do projeto:

O objetivo deste projeto é desenvolver um sistema eficaz e acessível de detecção de fumo usando o sensor MQ-135 e o ESP32. Este sistema será idealizado para monitorizar a qualidade do ar em tempo real e alertar os utilizadores quando níveis perigosos de fumo são detetados.

O sistema utilizará o sensor MQ-135 para detetar a presença de fumo e outros gases nocivos. Os dados do sensor serão processados pelo ESP32, que é um microcontrolador de baixo custo com capacidades Wi-Fi integradas. O ESP32 irá transmitir os dados do sensor para o Firebase, uma plataforma de desenvolvimento de aplicações na cloud, onde os dados podem ser armazenados e analisados.

Além disso, o sistema será construído para ser fácil de instalar e manter, tornando-o uma solução prática para residências, escritórios e outros espaços fechados.

Em última análise, o objetivo é criar um sistema de detecção de fumo que possa ajudar a proteger as pessoas e propriedades contra os perigos do fumo e melhorar a segurança e a saúde em ambientes fechados.

Metodologia

Descrição do sistema:

O sistema de detecção de fumo desenvolvido neste projeto é composto por vários componentes de hardware e software:

Hardware:

- **ESP32 WROOM 32D:** Este é o microcontrolador que escolhemos para o projeto, responsável por processar os dados do sensor MQ-135 e transmiti-los via Wi-Fi.
- **Breadboards:** São usadas para montar o circuito do sistema.
- **LEDs:** Três LEDs de cores diferentes (vermelho, verde e azul) são usados para indicar o estado do sistema.
- **Buzzers:** São usados para emitir um alarme sonoro quando o nível de fumo detetado excede um certo limite.
- **Resistências de 300 ohms:** São usadas no circuito para limitar a corrente que passa pelos LEDs.
- **Cabo jumpers:** São usados para conectar os vários componentes na breadboard.
- **Cabos mini USB:** São usados para alimentar o ESP32.
- **Powerbank:** É usada como fonte de alimentação para o sistema.

Software:

- **Arduino IDE:** É a plataforma usada para programar o ESP32. O código é escrito em C/C++ e carregado no ESP32 através da IDE do Arduino.
- **Firebase:** É uma plataforma de desenvolvimento de aplicações na nuvem fornecida pela Google. Neste projeto, é usada para armazenar e analisar os dados do sensor MQ-135 transmitidos pelo ESP32.
- **Jupyter Notebook:** O Jupyter Notebook é uma aplicação web de código aberto que permite criar e partilhar documentos interativos, chamados notebooks. Estes notebooks podem conter código ao vivo, equações, visualizações, texto narrativo e outros tipos de mídia.

Assim, o sistema funciona quando o sensor MQ-135 deteta a presença de fumo no ambiente. Esses dados são então processados pelo ESP32 e transmitidos para o Firebase via Wi-Fi. Se o nível de fumo detetado exceder um certo limite, o sistema aciona os LEDs e o buzzer para alertar os utilizadores. Os dados do sensor também podem ser monitorizados em tempo real através do Firebase.

Configuração dos sensores:

A configuração dos sensores no projeto envolve vários passos:

1. **Montagem do Circuito:** Primeiro, o sensor MQ-135 é conectado à breadboard. O pino VCC do sensor é conectado à fonte de alimentação (geralmente 5V), o pino GND é conectado à terra e o pino AO (analógico) é conectado a um pino analógico no ESP32. Os LEDs e o buzzer também são conectados à breadboard, com resistências de 300 ohms para limitar a corrente que passa pelos LEDs.
2. **Programação do ESP32:** O ESP32 é programado usando a IDE do Arduino. O código é escrito em C/C++, e é responsável por ler os dados do sensor MQ-135, processá-los e transmiti-los via Wi-Fi. O código também inclui lógica para acionar os LEDs e o buzzer quando o nível de fumo detetado excede um certo limite.
3. **Calibração do Sensor:** Antes de usar o sensor MQ-135, é necessário um tempo de pré-aquecimento de mais de 24 horas. Além disso, ao alimentar o sensor para detecção de gás, ele precisa de cerca de 60 a 120 segundos para se estabilizar. A calibração do sensor é um passo importante para garantir leituras precisas.
4. **Teste do Sistema:** Após a montagem e programação, o sistema é testado para garantir que está a funcionar corretamente. Isso inclui verificar se o sensor está a detetar corretamente o fumo e se os dados estão a ser transmitidos corretamente para o Firebase.

Transmissão de dados:

A transmissão de dados neste projeto é realizada através de uma conexão Wi-Fi entre o ESP32 e o Firebase. O ESP32 possui capacidades Wi-Fi integradas, o que permite transmitir dados diretamente para a Internet sem a necessidade de um microcontrolador separado.

O processo de transmissão de dados é o seguinte:

1. **Leitura de Dados:** O ESP32 lê os dados do sensor MQ-135. Estes dados representam a quantidade de fumo detetada pelo sensor.
2. **Processamento de Dados:** O ESP32 processa os dados do sensor. Isto pode incluir a conversão dos dados brutos do sensor numa medida útil, como a concentração de fumo no ar.
3. **Transmissão de Dados:** O ESP32 transmite os dados processados para o Firebase através de uma conexão Wi-Fi. Isto é feito usando a biblioteca Firebase do ESP32, que permite enviar dados diretamente para uma base de dados Firebase.
4. **Receção de Dados:** O Firebase recebe os dados transmitidos e armazena-os numa base de dados. Estes dados podem ser acedidos e analisados através da plataforma Firebase.
5. **Armazenamento de Dados:** O Jupyter Notebook permite-nos uma versatilidade na forma como podemos tratar os nossos dados.

Tentativas Práticas

- **Tentativa 1 (Arduino Uno)**

Na primeira tentativa, utilizamos o Arduino como uma ferramenta de teste para explorar a possibilidade de conectar e coletar dados do sensor MQ-135 através do ESP32. O objetivo principal deste experimento foi entender a funcionalidade e a eficácia do sensor MQ-135 quando integrado com o ESP32, utilizando o Arduino como interface. Este procedimento foi realizado exclusivamente para fins de teste, a fim de garantir que os componentes selecionados fossem adequados e funcionassem conforme esperado para o projeto em questão. Esta informação é crucial para a minha tese, pois fornece uma base sólida para futuras experimentações e implementações no campo da coleta de dados de sensores.

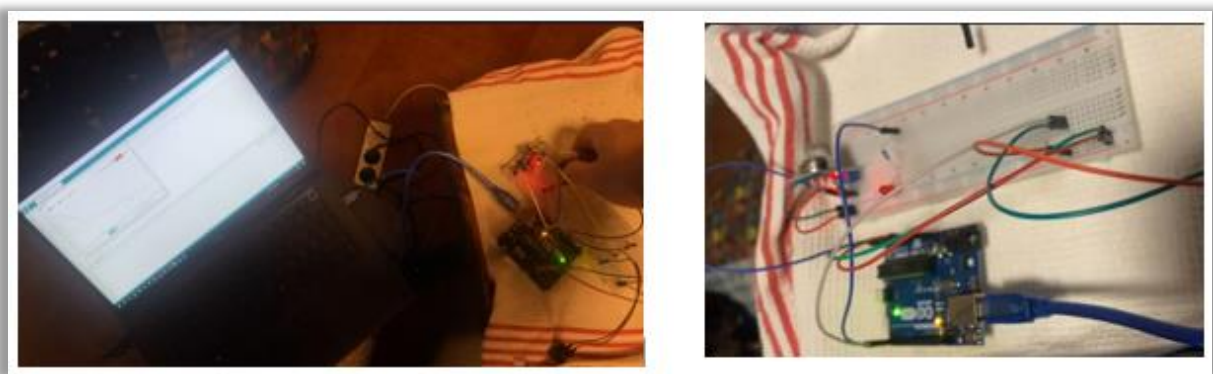


Figura 1 - Arduino com MQ135

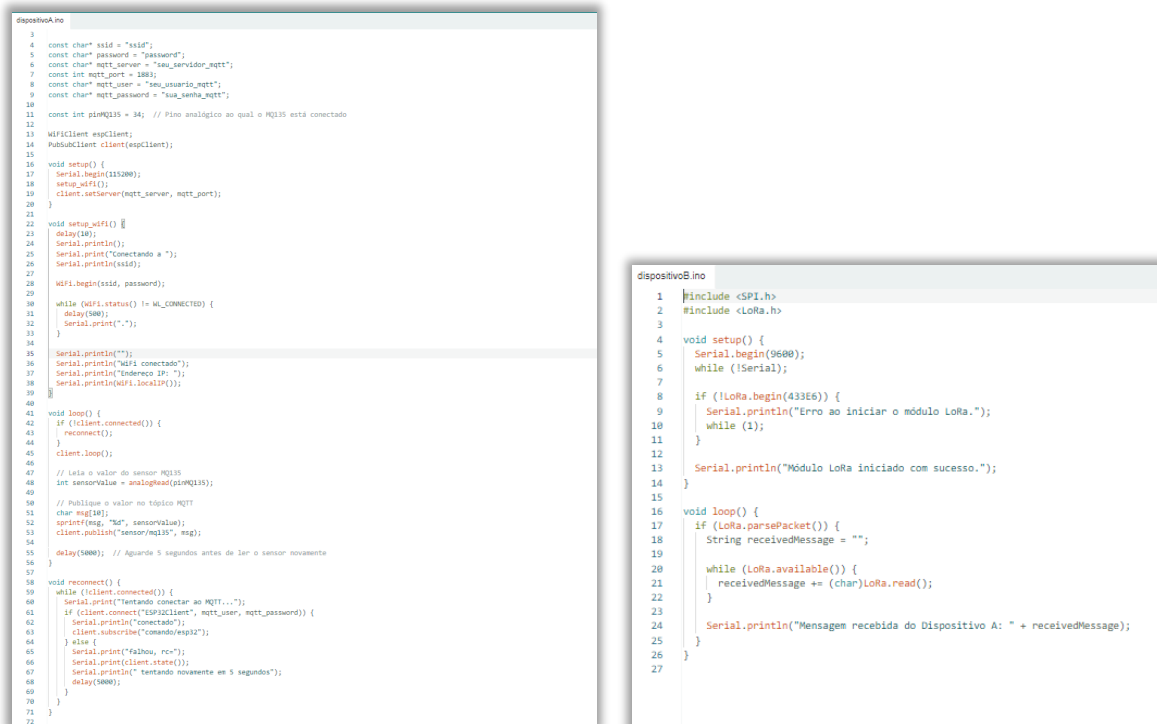
```
sensor1.ino
1  int sensorValue;
2  int digitalValue;
3
4  void setup()
5  {
6    Serial.begin(115200); //baud rate
7    pinMode(5, OUTPUT);
8    pinMode(18, OUTPUT);
9
10 }
11
12 void loop()
13 {
14
15   int sensorValue = analogRead(34);
16   int digitalValue = digitalRead(35);
17   if (sensorValue > 400)
18   {
19     Serial.println("Perigo de incêndio");
20     Serial.println("Alerte a Proteção civil, ligue 112");
21     Serial.println();
22     Serial.println();
23     digitalWrite(5, HIGH);
24     tone(18, 1000);
25     delay(50);
26     noTone(18);
27   }
28   else
29   {
30     Serial.println("Valores dentro do expectavel");
31     Serial.println(sensorValue, DEC);
32     //Serial.println(digitalValue, DEC);
33     digitalWrite(5, LOW);
34     digitalWrite(18, LOW);
35
36     delay(1000);
37   }
38 }
```

Figura 2 - Código Arduino IDE

- **Tentativa 2 (LoRa)**

Na segunda tentativa, optamos por utilizar dois dispositivos TTGO equipados com a tecnologia LoRa. Deve-se a escolha desta tecnologia à sua capacidade de permitir a comunicação de longo alcance entre dispositivos, o que seria benéfico para o nosso projeto.

Apesar de termos conseguido estabelecer uma comunicação bem-sucedida entre os dois dispositivos TTGO, encontramos algumas dificuldades na partilha de dados. Estas dificuldades podem ter sido causadas por vários fatores, incluindo possíveis limitações na capacidade de transmissão de dados da tecnologia LoRa, problemas de compatibilidade entre os dispositivos e questões relacionadas com a configuração do software.



```

dispositivoA.ino
3
4 const char* ssid = "ssid";
5 const char* password = "password";
6 const char* mqtt_server = "osu.servidor_mqtt";
7 const int mqtt_port = 1883;
8 const char* mqtt_user = "osu_usuario_mqtt";
9 const char* mqtt_password = "osu_senha_mqtt";
10
11 const int pinMQ135 = 34; // Pino analógico ao qual o MQ135 está conectado
12
13 WiFiClient espClient;
14 PubSubClient client(espClient);
15
16 void setup() {
17   Serial.begin(115200);
18   setup_wifi();
19   client.setServer(mqtt_server, mqtt_port);
20 }
21
22 void setup_wifi() {
23   delay(10);
24   Serial.println();
25   Serial.print("Conectando a ");
26   Serial.println(ssid);
27
28   WiFi.begin(ssid, password);
29
30   while (WiFi.status() != WL_CONNECTED) {
31     delay(500);
32     Serial.print(".");
33   }
34
35   Serial.println("");
36   Serial.println("WiFi conectado");
37   Serial.println("Endereço IP: ");
38   Serial.println(WiFi.localIP());
39 }
40
41 void loop() {
42   if (!client.connected()) {
43     reconnect();
44   }
45   client.loop();
46
47   // Lê o valor do sensor MQ135
48   int sensorValue = analogRead(pinMQ135);
49
50   // Publique o valor no tópico MQTT
51   char msg[10];
52   sprintf(msg, "%d", sensorValue);
53   client.publish("sensor/mq135", msg);
54   delay(5000); // Aguarde 5 segundos antes de ler o sensor novamente
55 }
56
57 void reconnect() {
58   while (!client.connected()) {
59     Serial.print("Tentando conectar ao MQTT...");
60     if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
61       Serial.println("conectado");
62       client.subscribe("comando/esp32");
63     } else {
64       Serial.print("falhou, rc=");
65       Serial.println(client.state());
66       Serial.println("Tentando novamente em 5 segundos");
67       delay(5000);
68     }
69   }
70 }
71 }
72
dispositivoB.ino
1 #include <SPI.h>
2 #include <LoRa.h>
3
4 void setup() {
5   Serial.begin(9600);
6   while (!Serial);
7
8   if (!LoRa.begin(433E6)) {
9     Serial.println("Erro ao iniciar o módulo LoRa.");
10    while (1);
11  }
12
13  Serial.println("Módulo LoRa iniciado com sucesso.");
14 }
15
16 void loop() {
17   if (LoRa.parsePacket()) {
18     String receivedMessage = "";
19
20     while (LoRa.available()) {
21       receivedMessage += (char)LoRa.read();
22     }
23
24     Serial.println("Mensagem recebida do Dispositivo A: " + receivedMessage);
25   }
26 }
27

```

Figura 3- Código LoRa

Deste modo, não nos pareceu ser uma alternativa viável para o tipo de projecto que idealizamos implementar.

- **Tentativa 3 (BLE – Bluetooth Low Energy)**

Na terceira tentativa, optamos por utilizar a tecnologia Bluetooth Low Energy (BLE) para estabelecer uma conexão entre dois dispositivos ESP32 e um terceiro dispositivo que atuaria como servidor. A escolha do BLE foi motivada pela sua eficiência energética e capacidade de facilitar a comunicação entre dispositivos em proximidade próxima.

No entanto, durante os nossos testes, deparamo-nos com um desafio significativo: embora conseguíssemos transmitir dados de cada dispositivo ESP32 para o servidor, não conseguíamos fazê-lo de ambos os dispositivos simultaneamente. Esta limitação representou um obstáculo para o nosso projeto, pois a nossa meta era coletar e analisar dados de múltiplos dispositivos em tempo real.

Após avaliar cuidadosamente os resultados desta tentativa, concluímos que a tecnologia BLE, na sua forma atual, não seria viável para o nosso projeto

```

BLEServer.ino
1 #include <BLEDevice.h>
2 #include <BLEServer.h>
3 #include <BLEUtils.h>
4 #include <BLE2902.h>
5
6 BLECharacteristic *pCharacteristic;
7 bool deviceConnected = false;
8 float txValue = 0;
9
10 #define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
11 #define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea87765b26a8"
12
13 class MyServerCallbacks: public BLEServerCallbacks {
14     void onConnect(BLEServer* pServer) {
15         deviceConnected = true;
16     };
17     void onDisconnect(BLEServer* pServer) {
18         deviceConnected = false;
19     };
20 };
21
22 void setup() {
23     Serial.begin(115200);
24
25     // Create the BLE Device
26     BLEDevice::init("ESP32_Mroom_032");
27
28     // Create the BLE Server
29     BLEServer *pServer = BLEDevice::createServer();
30     pServer->setCallbacks(new MyServerCallbacks());
31
32     // Create the BLE Service
33     BLEService *pService = pServer->createService(SERVICE_UUID);
34
35     // Create a BLE Characteristic
36     pCharacteristic = pService->createCharacteristic(
37         CHARACTERISTIC_UUID,
38         BLECharacteristic::PROPERTY_READ |
39         BLECharacteristic::PROPERTY_WRITE |
40         BLECharacteristic::PROPERTY_NOTIFY |
41         BLECharacteristic::PROPERTY_INDICATE
42     );
43
44     // Create a BLE Descriptor
45     pCharacteristic->addDescriptor(new BLE2902());
46
47     // Start the service
48     pService->start();
49
50     // Start advertising
51     pServer->getAdvertising()->start();
52     Serial.println("Waiting for a client connection to notify...");
53
54 }
55
56 void loop() {
57     if (deviceConnected) {
58         pCharacteristic->setValue((uint8_t*)&txValue, sizeof(txValue));
59         pCharacteristic->notify();
60         txValue += 0.01;
61         if (txValue > 100) {
62             txValue = 0;
63         }
64         delay(10); // bluetooth stack will go into congestion, if too many packets are sent
65     }
66 }
    
```

```

BLEClient.ino
1 #include <BLEDevice.h>
2 #include <BLEClient.h>
3 #include <BLEUtils.h>
4 #include <BLE2902.h>
5
6 BLERemoteCharacteristic *pCharacteristic;
7 bool deviceConnected = false;
8 float txValue = 0;
9
10 #define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
11 #define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea87765b26a8"
12
13 class MyClientCallbacks: public BLEClientCallbacks {
14     void onConnect(BLEClient* pClient) {
15         deviceConnected = true;
16     };
17     void onDisconnect(BLEClient* pClient) {
18         deviceConnected = false;
19     };
20 };
21
22 void setup() {
23     Serial.begin(115200);
24
25     // Create the BLE Device
26     BLEDevice::init("ESP32_Mroom_032");
27
28     // Connect to the BLE Server
29     BLEClient *pClient = BLEDevice::createClient();
30     pClient->setClientCallbacks(new MyClientCallbacks());
31     BLEAddress serverAddress("00:00:00:00:00:00");
32     pClient->connect(serverAddress);
33
34     // Obtain a reference to the service we are after in the remote BLE server.
35     BLERemoteServiceReference pRemoteServiceRef = pClient->getService(SERVICE_UUID);
36     if (pRemoteServiceRef == nullptr) {
37         Serial.println("Failed to find our service UUID: ");
38         Serial.println(SERVICE_UUID);
39         pClient->disconnect();
40         return;
41     }
42
43     // Obtain a reference to the characteristic in the service of the remote BLE server.
44     BLERemoteCharacteristic pRemoteCharacteristic = pRemoteServiceRef->getCharacteristic(CHARACTERISTIC_UUID);
45     if (pRemoteCharacteristic == nullptr) {
46         Serial.println("Failed to find our characteristic UUID: ");
47         Serial.println(CHARACTERISTIC_UUID);
48         pClient->disconnect();
49         return;
50     }
51
52     // Read the value of the characteristic.
53     std::string value = pRemoteCharacteristic->readValue().c_str();
54     Serial.print("The characteristic value was: ");
55     Serial.println(value.c_str());
56
57     // Subscribe to the characteristic
58     pRemoteCharacteristic->registerForNotify();
59     Serial.print("Received notification: ");
60     for (int i = 0; i < length; i++) {
61         Serial.print(data[i]);
62     }
63     Serial.println();
64
65     // Start the service.
66     pRemoteServiceRef->start();
67
68     Serial.println("Waiting for notifications...");
69
70 }
71
72 void loop() {
73     if (deviceConnected) {
74         pCharacteristic->writeValue((uint8_t*)&txValue, sizeof(txValue));
75         txValue += 0.01;
76         if (txValue > 100) {
77             txValue = 0;
78         }
79         delay(10); // bluetooth stack will go into congestion, if too many packets are sent
80     }
81 }
    
```

Figura 4 - Código BLE (Bluetooth Low Energy)

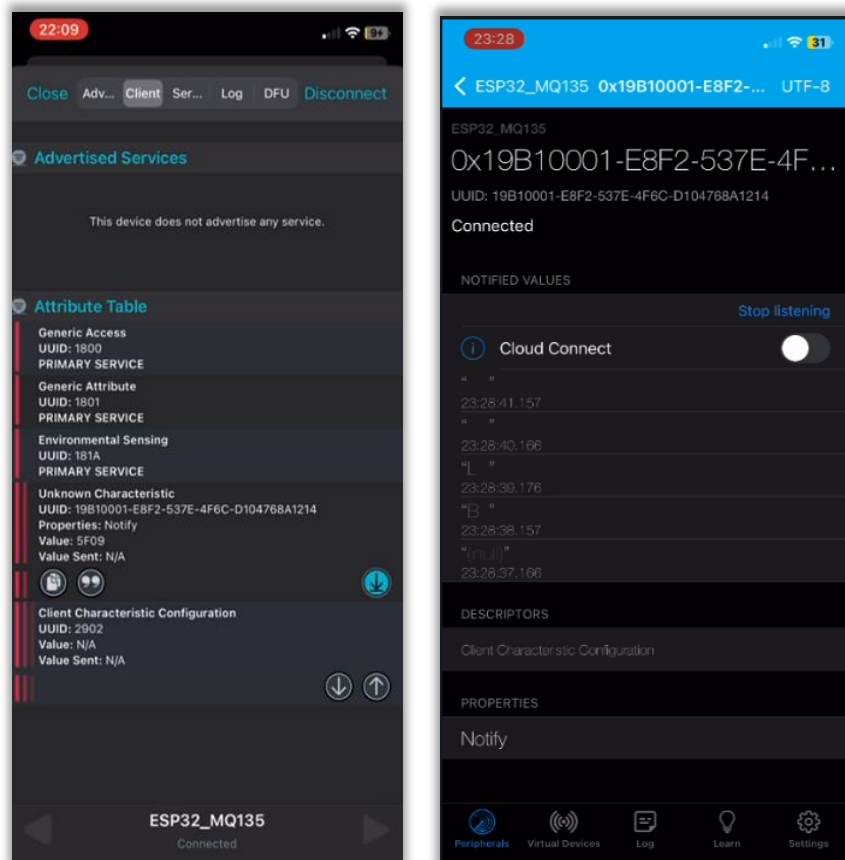


Figura 5 - Leitura de Dados na Aplicação nRF

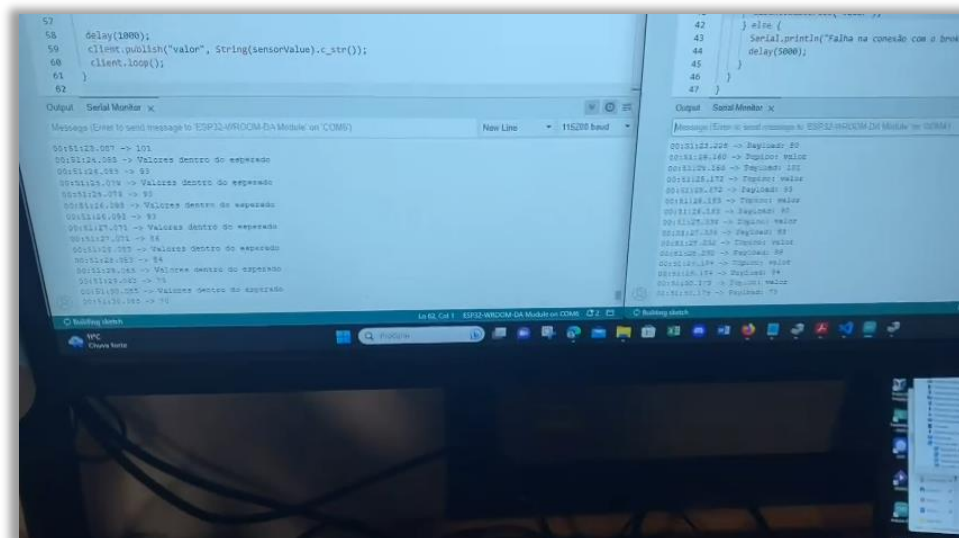


Figura 6 - ESP32 Rx<>Tx por BLE

- **Tentativa 4 MQTT**

Na quarta tentativa, decidimos implementar o protocolo MQTT (Message Queuing Telemetry Transport). A nossa intenção era utilizar o MQTT para facilitar a comunicação e a troca de dados entre os dispositivos do nosso sistema.

No entanto, durante a implementação, deparamo-nos com um desafio significativo. Embora tenhamos conseguido implementar o MQTT e transmitir dados através deste protocolo, tivemos dificuldades em compreender como poderíamos aproveitar e processar eficazmente os dados recebidos. Esta dificuldade pode ter sido causada por uma falta de familiaridade com o protocolo MQTT ou por uma falta de clareza sobre como os dados do MQTT poderiam ser integrados e utilizados no contexto do nosso projeto.

```
servidormqtt.ino
3
4 const char "ssid = "Homeland";
5 const char "password = "benficamada";
6 const char "mqtt_server = "192.168.1.34";
7 WiFiClient espClient;
8 PubSubClient client(espClient);
9 PubSubClient client2(espClient);
10
11 void setup() {
12   Serial.begin(115200);
13
14   // Conectar-se à rede WiFi
15   WiFi.begin(ssid, password);
16   while (WiFi.status() != WL_CONNECTED) {
17     delay(1000);
18     Serial.println("Conectando ao WiFi...");
19   }
20
21   // Configurar o cliente MQTT
22   client.setServer(mqtt_server, 1883);
23   client.setCallback(callback);
24   client2.setServer(mqtt_server, 1883);
25   client2.setCallback(callback);
26
27   // Subscriver aos tópicos específicos para cada dispositivo
28   client.subscribe("dispositivo/ESP32_Emissor_1/valor");
29   client2.subscribe("dispositivo/ESP32_Emissor_2/valor");
30 }
31
32 void callback(char "topic, byte "payload, unsigned int length) {
33   Serial.print("Tópico: ");
34   Serial.println(topic);
35
36   Serial.print("Payload: ");
37   for (int i = 0; i < length; i++) {
38     Serial.print((char)payload[i]);
39   }
40   Serial.println();
41 }
42
43 void reconnect() {
44   // Reconectar para o primeiro dispositivo
45   while (!client.connected()) {
46     Serial.println("Conectando ao broker MQTT (Dispositivo 1)...");
47     if (client.connect("ESP32_Receptor")) {
48       Serial.println("Conectado ao broker MQTT (Dispositivo 1)");
49       client.subscribe("dispositivo/ESP32_Emissor_1/valor");
50     } else {
51       Serial.println("Falha na conexão com o broker MQTT (Dispositivo 1). Tente novamente em 5 segundos...");
52       delay(5000);
53     }
54   }
55
56   // Reconectar para o segundo dispositivo
57   while (!client2.connected()) {
58     Serial.println("Conectando ao broker MQTT (Dispositivo 2)...");
59     if (client2.connect("ESP32_Receptor")) {
60       Serial.println("Conectado ao broker MQTT (Dispositivo 2)");
61       client2.subscribe("dispositivo/ESP32_Emissor_2/valor");
62       client2.subscribe("dispositivo/ESP32_Emissor_1/valor");
63     } else {
64       Serial.println("Falha na conexão com o broker MQTT (Dispositivo 2). Tente novamente em 5 segundos...");
65       delay(5000);
66     }
67   }
68 }
69
70 void loop() {
71   if (!client.connected()) {
72     reconnect();
73   }
74   client.loop();
75
76   if (!client2.connected()) {
77     reconnect();
78   }
79   client2.loop();
80
81   delay(1000); // Aguarda 1 segundo entre as iterações
82 }
83 }
```

Figura 7 - Código Servidor MQTT

```

1 #include <WiFi.h>
2 #include <Firebase_ESP_Client.h>
3 #include "soc/rtc.h" //Brownoutdetector disable
4 #include "soc/rtc_cntl_reg.h" //Brownoutdetector disable
5 //Provide the token generation process info.
6 #include "addons/TokenerHelper.h"
7 //Provide the RTDB payload printing info and other helper functions.
8 #include "addons/RTDBHelper.h"
9
10 // Insert your network credentials
11 #define WIFI_SSID "Homeland"
12 #define WIFI_PASSWORD "benficamerda"
13
14 // Insert Firebase project API Key
15 #define API_KEY "AIzaSyABXKPcSVT46wR3r_A3wMFFc5x2XNM"
16
17 // Insert RTDB URL/define the RTDB URL */
18 #define DATABASE_URL "https://iotmq13step2-default-rtdb.europe-west1.firebaseio.com/"
19
20 //Definir variáveis porta digital e porta analógico
21 int sensorValue;
22 int digitalValue;
23
24 //Define Firebase Data object
25 FirebaseData fdbdo;
26
27 FirebaseAuth auth;
28 FirebaseConfig config;
29
30 unsigned long sendDataPrevMillis = 0;
31 int count = 0;
32 bool signupOK = false;
33
34 void disableBrownoutDetector() {
35   WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // Desabilita o Brownout Detector
36 }
37
38 void setup() {
39   //setCPUfrequency(80);
40   Serial.begin(115200);
41   disableBrownoutDetector();
42   pinMode(34, INPUT); // Pino analógico para o sensor
43   pinMode(35, INPUT); // Pino digital para o sensor
44   pinMode(5, OUTPUT); // Pino para o LED
45   pinMode(18, OUTPUT); // Pino para o Buzzer
46
47   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
48   Serial.print("Connecting to Wi-Fi");
49   while (WiFi.status() != WL_CONNECTED) {
50     Serial.print(".");
51     delay(3000);
52   }
53   Serial.println();
54   Serial.print("Connected with IP: ");
55   Serial.println(WiFi.localIP());
56   Serial.println();
57
58   /* Assign the api key (required) */
59   config.api_key = API_KEY;
60
61   /* Assign the RTDB URL (required) */
62   config.database_url = DATABASE_URL;
63
64   if (Firebase.signup(&config, &auth, "", "")) {
65     Serial.println("ok");
66     signupOK = true;
67   } else {
68     Serial.println("Error", config.signer.signuperror.message.c_str());
69   }
70
71   // Assign the callback function for the long running token generation task */
72   config.token_status_callback = tokenstatuscallback; //see addons/TokenerHelper.h
73
74   Firebase.begin(&config, &auth);
75   Firebase.reconnectWiFi(true);
76
77   void loop() {
78     String alerta = "Perigo de Incêndio";
79
80     // Ler os valores dos sensores
81     int sensorValue = analogRead(34);
82     int digitalValue = digitalRead(35);
83
84     // Lógica de verificação do sensor de incêndio
85     if (sensorValue > 400) {
86       Serial.println("Perigo de Incêndio");
87       Serial.println("Alerta a Protecção civil, ligue 112");
88       Serial.println();
89       digitalWrite(LED, HIGH);
90       tone(18, 1000);
91       delay(500);
92       noTone(18);
93
94       // Enviar dados para o Firebase em caso de perigo de incêndio
95       if (Firebase.ready() && signupOK) {
96         // Enviar valor do sensor analógico
97         Firebase.RTDB.setint(&fdbdo, "Salal/Valor CO2", sensorValue);
98
99         // Converte a String para um array de caracteres (const char*)
100         const char* alertaCharArray = alerta.c_str();
101
102         // Envia o array de caracteres para o Firebase
103         Firebase.RTDB.setString(&fdbdo, "Salal/Mensagem", alertaCharArray);
104       }
105     } else {
106       Serial.println("Valores dentro do expectável");
107       Serial.println(sensorValue, DEC);
108       //Serial.println(digitalValue, DEC);
109       digitalWrite(LED, LOW);
110
111       if (Firebase.ready() && signupOK) {
112         String alerta = "Valores dentro do expectável";
113         // Enviar valor do sensor analógico
114         Firebase.RTDB.setint(&fdbdo, "Salal/Valor CO2", sensorValue);
115
116         // Converte a String para um array de caracteres (const char*)
117         const char* alertaCharArray = alerta.c_str();
118
119         // Envia o array de caracteres para o Firebase
120         Firebase.RTDB.setString(&fdbdo, "Salal/Mensagem", alertaCharArray);
121       }
122     }
123     delay(1000);
124   }
125 }

```

Figura 8 - Código Cliente MQTT

- Tentativa 5 Dashboard

Na quinta tentativa, voltamos a nossa atenção para o desenvolvimento de um Dashboard para o nosso projeto. O objetivo era criar uma interface visual que nos permitisse monitorizar e analisar os dados coletados pelos nossos dispositivos de forma eficiente e intuitiva.

Exploramos várias abordagens para alcançar este objetivo. Entre as opções consideradas estavam a integração do Firebase com o Node-RED, o uso do Flutter, Blynk e a utilização do ThingSpeak. Cada uma destas opções apresentava as suas próprias vantagens e desvantagens, e todas requeriam um certo nível de conhecimento técnico para a sua implementação.

No entanto, após uma avaliação cuidadosa, chegamos à conclusão de que, dadas as nossas limitações de tempo e o nosso nível atual de conhecimento sobre estas tecnologias, não seria viável prosseguir com nenhuma destas abordagens neste momento.

- **Discussão**

Os resultados das tentativas que não foram bem-sucedidas fornecem informações valiosas para o projeto e são tão importantes quanto os sucessos. Aqui está uma breve interpretação dos resultados:

1. **Arduino e ESP32 com sensor MQ-135:** A dificuldade em extrair e processar os dados do sensor MQ-135 indica que pode ser necessário um maior entendimento do funcionamento do sensor ou uma abordagem diferente para a coleta de dados.
2. **Comunicação LoRa entre dispositivos TTGO:** A dificuldade em partilhar dados entre os dispositivos sugere que a tecnologia LoRa pode ter limitações na transmissão de dados ou que pode haver problemas de compatibilidade entre os dispositivos.
3. **Conexão BLE entre ESP32:** A incapacidade de transmitir dados de ambos os dispositivos ESP32 simultaneamente para o servidor indica que a tecnologia BLE pode não ser adequada para aplicações que requerem a transmissão simultânea de dados de múltiplos dispositivos.
4. **Implementação do MQTT:** A dificuldade em aproveitar e processar os dados transmitidos através do MQTT sugere que pode ser necessário um maior conhecimento do protocolo MQTT e uma melhor compreensão de como os dados do MQTT podem ser utilizados eficazmente.
5. **Desenvolvimento do Dashboard:** A dificuldade em implementar o Firebase, Node-RED, Flutter e ThingSpeak para o desenvolvimento do Dashboard sugere que pode ser necessário mais tempo e conhecimento para implementar eficazmente estas tecnologias.

Em cada uma destas tentativas, embora os resultados não tenham sido os esperados, foram obtidos insights valiosos que podem informar e orientar futuras decisões e estratégias do projeto.

Elaboração do Projeto

Escolhemos a seguinte solução para a implementação final do nosso projeto, que envolveu uma série de etapas detalhadas e cuidadosamente planejadas. ESP32 WROOM 32D: Este é o microcontrolador que escolhemos para o projeto, responsável por processar os dados do sensor MQ-135 e transmiti-los via Wi-Fi.

Material utilizado

- Breadboards;
- LEDs;
- Buzzer;
- Resistências de 300 ohms;
- Cabo jumpers;
- Cabos mini USB.

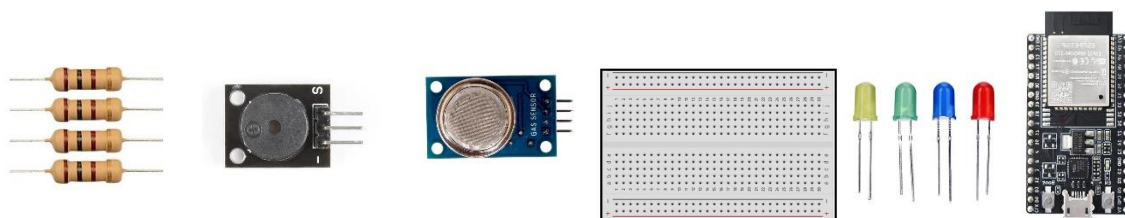


Figura 9 - Material utilizado

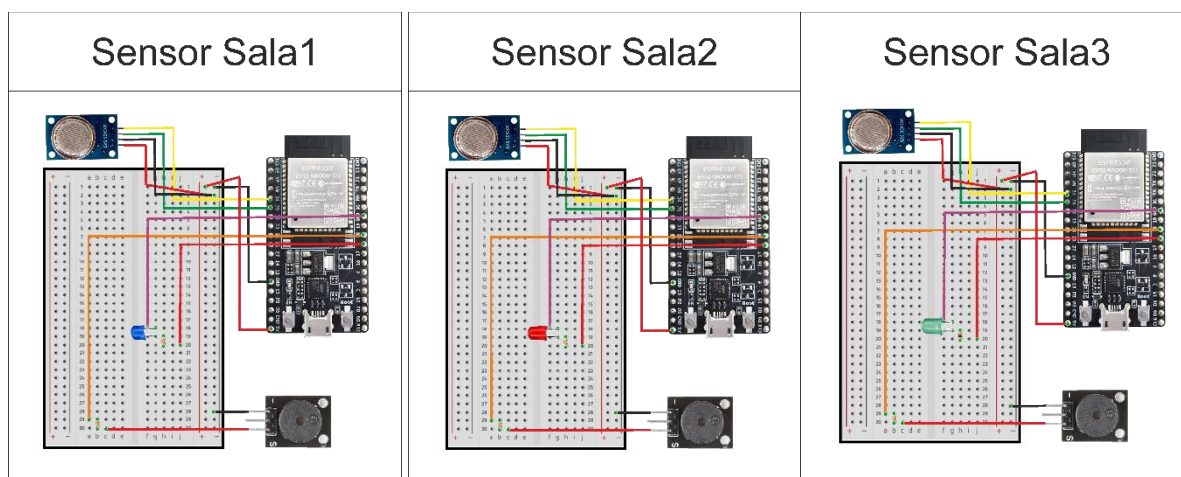


Figura 10 - Esquema dos 3 dispositivos elaborados

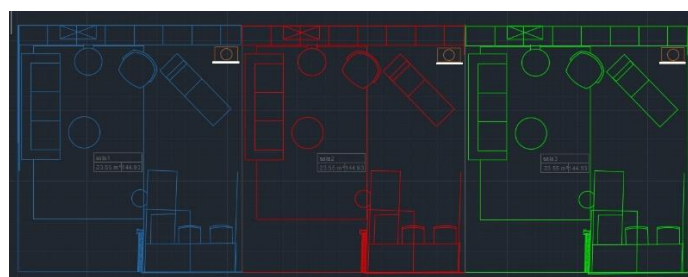


Figura 11 - As 3 salas de teste dos sensores

Explicação do Código Arduino IDE

Inclusão de bibliotecas: as bibliotecas necessárias para o funcionamento do programa são incluídas. Isso inclui a biblioteca WiFi.h para a conectividade Wi-Fi, Firebase_ESP_Client.h para a interação com o Firebase, e outras bibliotecas para várias funcionalidades.

```
espsala1.ino
1  #include <WiFi.h>
2  #include <Firebase_ESP_Client.h>
3  #include "soc/soc.h" //Brownoutdetector disable
4  #include "soc/rtc_cntl_reg.h" //Brownoutdetector disable
5  //Provide the token generation process info.
6  #include "addons/TokenHelper.h"
7  //Provide the RTDB payload printing info and other helper functions.
8  #include "addons/RTDBHelper.h"
9
```

Figura 12 – Inclusão das Bibliotecas

Definição de constantes: As constantes para as credenciais da rede Wi-Fi, a chave da API do Firebase e a URL do banco de dados em tempo real (RTDB) são definidas.

```
// Insert your network credentials
#define WIFI_SSID "Homeland"
#define WIFI_PASSWORD "benficamerda"

// Insert Firebase project API Key
#define API_KEY "AIzaSyAbXTXPc5VT46wRJR__AjwhMfPcr5xdJKM"

// Insert RTDB URLdefine the RTDB URL */
#define DATABASE_URL "https://iotmq135esp32-default-rtdb.europe-west1.firebaseio.com/"
```

Figura 13 - Definição de Constantes

Definição de variáveis: As variáveis sensorValue e digitalValue são definidas para armazenar os valores lidos dos sensores. Vários objetos para interagir com o Firebase também são definidos.

```
19
20 //Definir variaveis porta digital e porta analógico
21 int sensorValue;
22 int digitalValue;
23
```

Figura 14 - Definição de Variáveis

Função disableBrownoutDetector: Esta função desativa o detetor de brownout, que é um recurso de segurança que reinicia o ESP32 se a tensão de alimentação cair abaixo de um certo nível.

```
34 void disableBrownoutDetector() {  
35     WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // Desabilita o Brownout Detector  
36 }  
37
```

Figura 15 - Função disableBrownoutDetector

Função setup: esta função é chamada apenas quando o programa começa. Esta configura a frequência da CPU, inicia a comunicação serial, desativa o detetor de brownout, configura os pinos de entrada e saída, conecta-se à rede Wi-Fi e configura a conexão com o Firebase.

```
39 void setup() {  
40     setCpuFrequencyMhz(80);  
41     Serial.begin(115200);  
42     disableBrownoutDetector();  
43     pinMode(34, INPUT); // Pino analógico para o sensor  
44     pinMode(35, INPUT); // Pino digital para o sensor  
45     pinMode(5, OUTPUT); // Pino para o LED  
46     pinMode(18, OUTPUT); // Pino para o Buzzer  
47  
48     WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
49     Serial.print("Connecting to Wi-Fi");  
50     while (WiFi.status() != WL_CONNECTED){  
51         Serial.print(".");  
52         delay(300);  
53     }  
54     Serial.println();  
55     Serial.print("Connected with IP: ");  
56     Serial.println(WiFi.localIP());  
57     Serial.println();  
58     /* Assign the api key (required) */  
59     config.api_key = API_KEY;  
60     /* Assign the RTDB URL (required) */  
61     config.database_url = DATABASE_URL;  
62     /* Sign up */  
63     if (Firebase.signUp(&config, &auth, "", "")){  
64         Serial.println("ok");  
65         signupOK = true;  
66     }  
67     else{  
68         Serial.printf("%s\n", config.signer.signupError.message.c_str());  
69     }  
70     /* Assign the callback function for the long running token generation task */  
71     config.token_status_callback = tokenStatusCallback; //see addons/TokenHelper.h  
72     Firebase.begin(&config, &auth);  
73     Firebase.reconnectWiFi(true);  
74 }  
75  
76
```

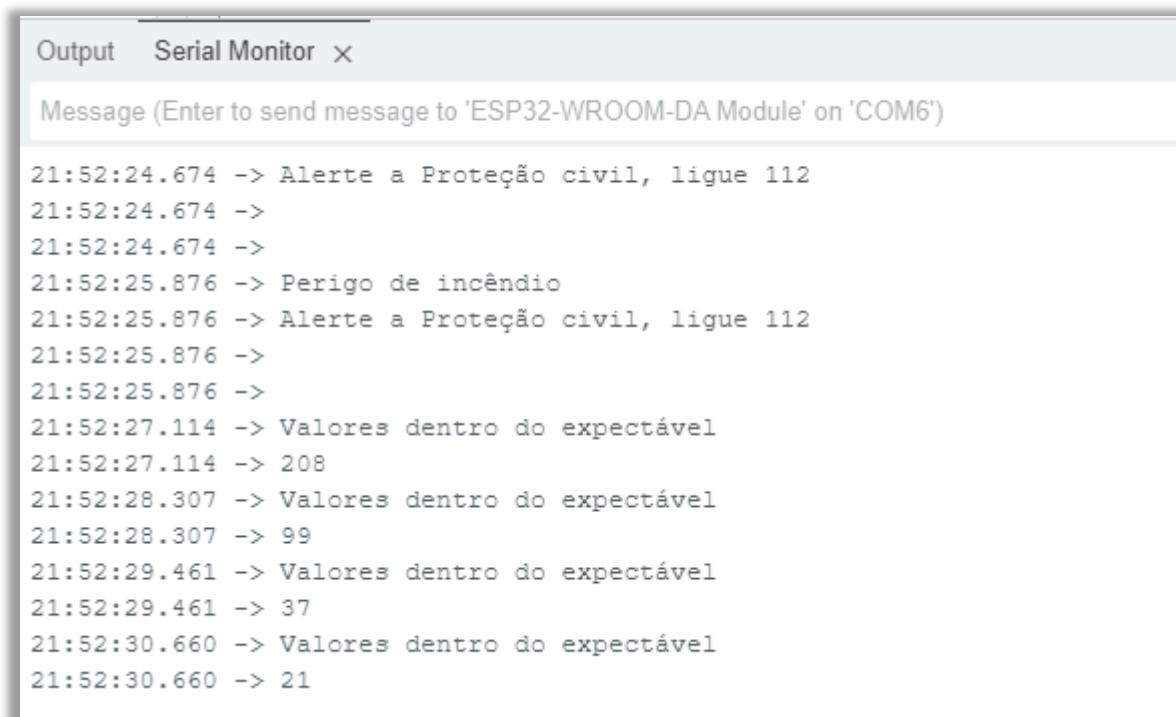
Figura 16 - Função Setup

Função loop: esta função é chamada repetidamente enquanto o programa está em execução. Esta lê os valores dos sensores, verifica se o valor do sensor analógico é maior que 400ppm (valor parametrizado para indicar um possível incêndio), e envia os dados para o Firebase. Se o valor do sensor analógico é menor ou igual a 400ppm, o programa envia uma mensagem diferente para o Firebase indicando que os valores estão dentro do esperado.

```
77 void loop(){
78   String alerta = "Perigo de Incendio";
79   int sensorValue = analogRead(34);
80   int digitalValue = digitalRead(35);
81   // Lógica de verificação do sensor de incêndio
82   if (sensorValue > 400){
83     Serial.println("Perigo de incêndio");
84     Serial.println("Alerte a Proteção civil, ligue 112");
85     Serial.println();
86     Serial.println();
87     digitalWrite(5, HIGH);
88     tone(18, 1000);
89     delay(50);
90     noTone(18);
91     // Enviar dados para o Firebase em caso de perigo de incêndio
92     if (Firebase.ready() && signupOK){
93       // Enviar valor do sensor analógico
94       Firebase.RTDB.setInt(&fbdo, "Sala1/Valor CO2", sensorValue);
95       // Converte a String para um array de caracteres (const char*)
96       const char* alertaCharArray = alerta.c_str();
97       // Envia o array de caracteres para o Firebase
98       Firebase.RTDB.setString(&fbdo, "Sala1/Mensagem", alertaCharArray);
99     }
100   }
101   else{
102     Serial.println("Valores dentro do expectável");
103     Serial.println(sensorValue, DEC);
104     //Serial.println(digitalValue, DEC);
105     digitalWrite(5, LOW);
106     //digitalWrite(18, LOW);
107     if (Firebase.ready() && signupOK){
108       String alerta = "Valores dentro do expectável";
109       // Enviar valor do sensor analógico
110       Firebase.RTDB.setInt(&fbdo, "Sala1/Valor CO2", sensorValue);
111       // Converte a String para um array de caracteres (const char*)
112       const char* alertaCharArray = alerta.c_str();
113       // Envia o array de caracteres para o Firebase
114       Firebase.RTDB.setString(&fbdo, "Sala1/Mensagem", alertaCharArray);
115     }
116   }
117   delay(1000);
118 }
```

Figura 17 - Função Loop

OUTPUT:



The screenshot shows a 'Serial Monitor' window with a title bar containing 'Output', 'Serial Monitor', and a close button 'X'. Below the title bar is a text input field with the placeholder 'Message (Enter to send message to 'ESP32-WROOM-DA Module' on 'COM6')'. The main area of the window displays a series of timestamped messages from a sensor. The messages are as follows:

```
21:52:24.674 -> Alerte a Proteção civil, ligue 112
21:52:24.674 ->
21:52:24.674 ->
21:52:25.876 -> Perigo de incêndio
21:52:25.876 -> Alerte a Proteção civil, ligue 112
21:52:25.876 ->
21:52:25.876 ->
21:52:27.114 -> Valores dentro do expectável
21:52:27.114 -> 208
21:52:28.307 -> Valores dentro do expectável
21:52:28.307 -> 99
21:52:29.461 -> Valores dentro do expectável
21:52:29.461 -> 37
21:52:30.660 -> Valores dentro do expectável
21:52:30.660 -> 21
```

Figura 18 - Output obtido no Sensor1

Criação de conta Firebase

Criamos uma conta no Firebase e configuramos a URL e a API para permitir a interação entre os nossos dispositivos e o banco de dados.

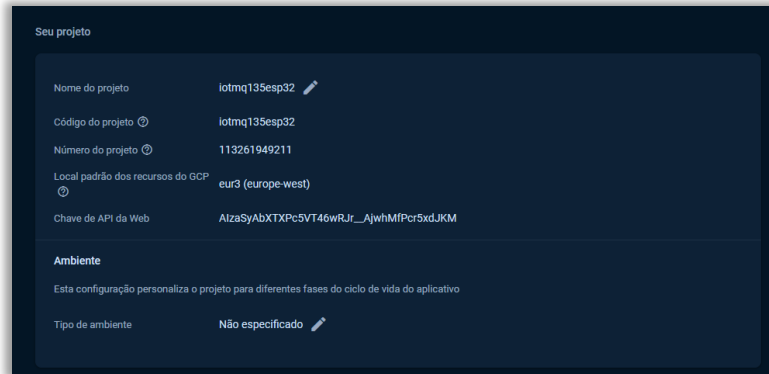


Figura 19 - Dados Projeto Firebase

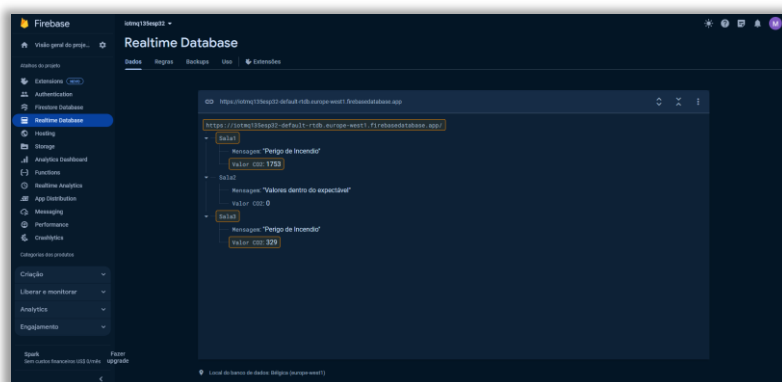


Figura 20 - Dados em RealTime Database

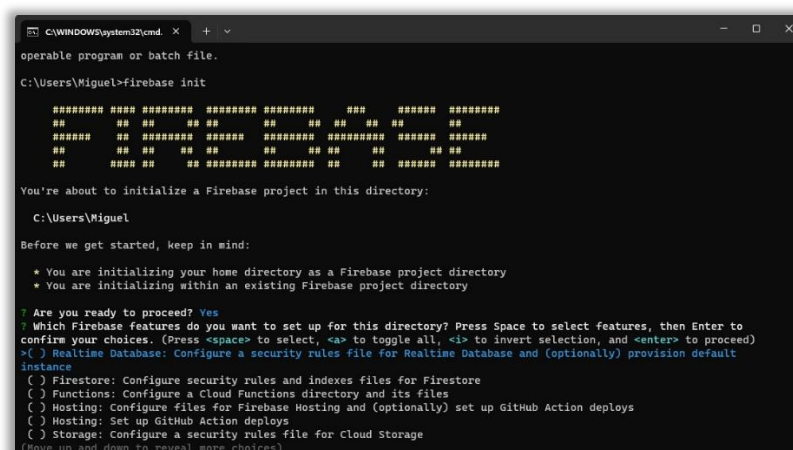
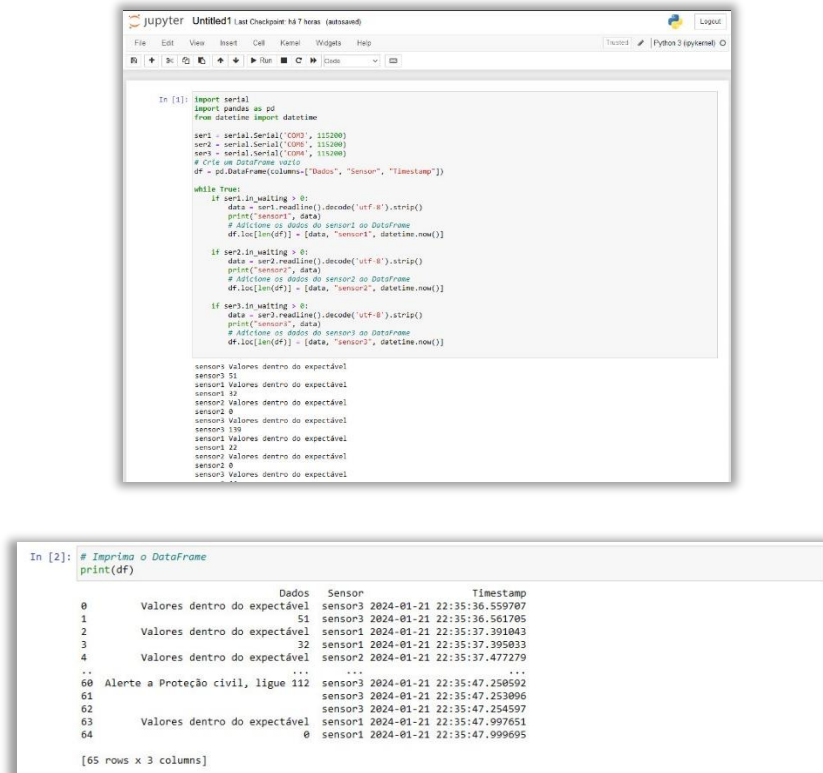


Figura 21 - Login e iniciação no Firebase Console

Acabamos de iniciar e fazer login na consola Firebase. Agora começamos a configurar e gerir o nosso projeto. Vamos começar a explorar as várias funcionalidades e serviços que o Firebase tem para oferecer, neste caso a base de dados em tempo real.

Armazenamento de dados

Optamos pelo Jupyter Notebook como a nossa plataforma de escolha para o armazenamento de dados. Durante as aulas de ciências de dados, percebemos que o Jupyter Notebook seria a opção mais adequada para as nossas necessidades, pois permite escolher o período que queremos exportar, exportar os dados para um arquivo CSV, e criar um gráfico onde podemos visualizar o histórico das medidas do sensor, entre outras funcionalidades.



```
In [1]: import serial
import pandas as pd
from datetime import datetime

ser1 = serial.Serial('/dev/ttyUSB0', 115200)
ser2 = serial.Serial('/dev/ttyUSB0', 115200)
ser3 = serial.Serial('/dev/ttyUSB0', 115200)
# Cria um DataFrame vazio
df = pd.DataFrame(columns=["Dados", "Sensor", "Timestamp"])

while True:
    if ser1.in_waiting > 0:
        data = ser1.readline().decode('utf-8').strip()
        print("sensor1", data)
        # Adicione os dados do sensor1 ao DataFrame
        df.loc[len(df)] = [data, "sensor1", datetime.now()]

    if ser2.in_waiting > 0:
        data = ser2.readline().decode('utf-8').strip()
        print("sensor2", data)
        # Adicione os dados do sensor2 ao DataFrame
        df.loc[len(df)] = [data, "sensor2", datetime.now()]

    if ser3.in_waiting > 0:
        data = ser3.readline().decode('utf-8').strip()
        print("sensor3", data)
        # Adicione os dados do sensor3 ao DataFrame
        df.loc[len(df)] = [data, "sensor3", datetime.now()]

sensor1 Valores dentro do expectável
sensor3 51
sensor1 Valores dentro do expectável
sensor1 12
sensor1 Valores dentro do expectável
sensor2 0
sensor1 Valores dentro do expectável
sensor1 139
sensor1 Valores dentro do expectável
sensor1 21
sensor1 Valores dentro do expectável
sensor2 0
sensor3 Valores dentro do expectável

In [2]: # Imprima o DataFrame
print(df)

      Dados  Sensor  Timestamp
0  Valores dentro do expectável  sensor3  2024-01-21 22:35:36.559787
1      51  sensor3  2024-01-21 22:35:36.561785
2  Valores dentro do expectável  sensor1  2024-01-21 22:35:37.391043
3      32  sensor1  2024-01-21 22:35:37.395033
4  Valores dentro do expectável  sensor2  2024-01-21 22:35:37.477279
...      ...      ...
60  Alerte a Proteção civil, ligue 112  sensor3  2024-01-21 22:35:47.250592
61      sensor3  2024-01-21 22:35:47.253096
62      sensor3  2024-01-21 22:35:47.254597
63  Valores dentro do expectável  sensor1  2024-01-21 22:35:47.997651
64      0  sensor1  2024-01-21 22:35:47.999695

[65 rows x 3 columns]
```

Figura 22 - Código Python a correr no Notebook

O código Python apresentado demonstra uma aplicação eficiente e eficaz para a recolha e armazenamento de dados de sensores em tempo real. Utilizando a biblioteca serial para comunicação com os sensores e a biblioteca pandas para manipulação e armazenamento de dados, o script é capaz de armazenar continuamente dados de três sensores diferentes e guardá-los de forma organizada num DataFrame. Além disso, a inclusão de carimbos de data/hora com cada leitura de sensor permite uma análise temporal detalhada dos dados coletados.

```
In [3]: # Exporte o DataFrame para um arquivo CSV
df.to_csv('dados_sensores.csv', index=False)
```

Figura 23 - Exportação dos dados para um ficheiro CSV

Este comando é útil para guardar dados em um formato que pode ser facilmente lido por outros programas, no nosso contexto o Excel.

```
In [4]: # Ler o arquivo CSV
dados = pd.read_csv('dados_sensores.csv')

# Imprimir o DataFrame
print(dados)
```

	Dados	Sensor	Timestamp
0	Valores dentro do expectável	sensor3	2024-01-21 22:35:36.559707
1	51	sensor3	2024-01-21 22:35:36.561705
2	Valores dentro do expectável	sensor1	2024-01-21 22:35:37.391043
3	32	sensor1	2024-01-21 22:35:37.395033
4	Valores dentro do expectável	sensor2	2024-01-21 22:35:37.477279
...
60	Alerte a Proteção civil, ligue 112	sensor3	2024-01-21 22:35:47.250592
61	NaN	sensor3	2024-01-21 22:35:47.253096
62	NaN	sensor3	2024-01-21 22:35:47.254597
63	Valores dentro do expectável	sensor1	2024-01-21 22:35:47.997651
64	0	sensor1	2024-01-21 22:35:47.999695

[65 rows x 3 columns]

Figura 24 - Leitura do ficheiro CSV no Notebook

Este código imprime o conteúdo do ficheiro no ecrã. O ficheiro 'dados_sensores.csv' é um ficheiro que contém dados de sensores que foram recolhidos anteriormente e guardados neste ficheiro.



Figura 25 - Criação de um Gráfico Dados/Tempo para cada sensor

Este código é utilizado para criar um gráfico dos dados de três sensores diferentes ao longo do tempo. Aqui está uma descrição passo a passo do que o código faz:

1. Importa as bibliotecas necessárias, neste caso, matplotlib.pyplot e pandas.
2. Supõe-se que 'df' é o DataFrame que contém os dados. A coluna 'dados' é convertida para numérico usando a função pd.to_numeric(). O argumento errors='coerce' é usado para lidar com quaisquer erros que possam ocorrer durante a conversão.

3. Em seguida, o código cria um gráfico para cada sensor único presente na coluna 'sensor' do DataFrame. Para cada sensor, filtra os dados correspondentes a esse sensor e traça os dados em função do tempo.
4. Adiciona títulos ao gráfico e aos eixos x e y usando as funções `plt.title()`, `plt.xlabel()` e `plt.ylabel()`.
5. Adiciona uma legenda ao gráfico com a função `plt.legend()`.
6. Finalmente, exibe o gráfico usando a função `plt.show()`.
7. Portanto, é uma forma eficiente de visualizar os dados de vários sensores ao longo do tempo num único gráfico.

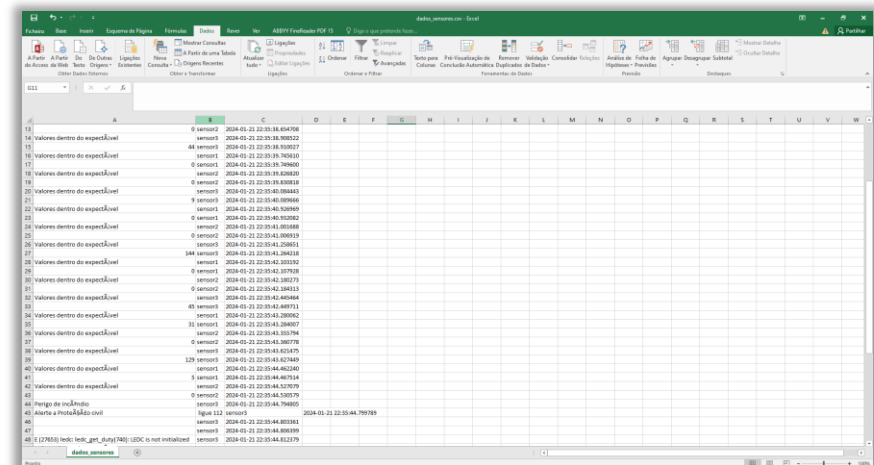


Figura 26 - Ficheiro CSV com dados exportados

Montagem de uma maquete

Para facilitar a compreensão da lógica do nosso projeto, montamos uma maquete. Esta maquete serviu como uma representação visual do nosso sistema, ajudando-nos a visualizar o seu funcionamento e a comunicar eficazmente a sua estrutura e design.



Figura 27 - Maquete do Projeto

Esta implementação final é um componente crucial, pois representa a culminação de várias tentativas. Esta ilustra a importância da persistência e adaptação no processo de desenvolvimento de um projeto de pesquisa. Além disso, destaca a importância de uma compreensão clara das tecnologias utilizadas e de uma implementação cuidadosa para garantir o sucesso do projeto.

Melhorias e direções futuras

Embora o sistema de detecção de fumo desenvolvido neste projeto tenha demonstrado ser eficaz, existem várias áreas em que poderia ser melhorado ou expandido no futuro:

1. **Calibração do Sensor:** A calibração do sensor MQ-135 pode ser um desafio, pois é afetada por vários fatores, incluindo a temperatura e a humidade. No futuro, poderíamos explorar métodos mais avançados de calibração para melhorar a precisão das leituras do sensor.
2. **Redução de Falsos Alarmes:** O sistema atual pode ser suscetível a falsos alarmes causados por interferências de outros gases. Poderíamos investigar métodos para reduzir a ocorrência de falsos alarmes, como o uso de algoritmos de aprendizagem de máquina para distinguir entre fumo e outros gases.
3. **Integração com Outros Sensores:** para aumentar a funcionalidade do sistema, poderíamos integrar outros sensores, como sensores de temperatura e humidade. Isso permitiria ao sistema monitorizar uma gama mais ampla de condições ambientais.
4. **Expansão para Outras Aplicações:** atualmente, o sistema é projetado principalmente para uso em ambientes fechados. No entanto, poderia ser adaptado para outras aplicações, como monitoramento da qualidade do ar ao ar livre ou detecção de gases tóxicos em ambientes industriais.
5. **Desenvolvimento de uma Interface de Utilizador:** para tornar o sistema mais fácil de usar, poderíamos desenvolver uma interface de utilizador amigável que permita aos utilizadores visualizar os dados do sensor em tempo real, ajustar as configurações do sistema e receber alertas quando os níveis de fumo excedem um certo limite.

Estas são apenas algumas das possíveis direções para o futuro desenvolvimento deste projeto. Acreditamos que há um grande potencial para expandir e melhorar o sistema.

Conclusão

Neste projeto, desenvolvemos um sistema eficaz e acessível de detecção de fumo usando o sensor MQ-135 e o ESP32. O sistema foi capaz de monitorizar a qualidade do ar em tempo real e alertar os utilizadores quando foram detetados níveis perigosos de fumo.

Os resultados obtidos indicam que o sistema é capaz de detetar fumo de forma eficaz e precisa. Além disso, a utilização do ESP32 permitiu a transmissão de dados em tempo real para o Firebase, facilitando a monitorização e a análise dos dados do sensor.

Comparado com outros sistemas de detecção de fumo disponíveis no mercado, o nosso sistema demonstrou várias vantagens, incluindo custo reduzido, facilidade de instalação e manutenção, e a capacidade de fornecer deteções precisas e confiáveis de fumo.

Em conclusão, este projeto demonstrou que é possível desenvolver um sistema de detecção de fumo eficaz e acessível usando tecnologias modernas como o sensor MQ-135 e o ESP32.