Relatório de Projeto

Rafael Mendes Marques

Engenharia Informática

dez | 2023

GUARDA
POLI
TÉCNICO

POLI TÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

Airsense
Qualidade do Ar Interior

PROJETO DE FIM DE CURSO
PARA OBTENÇÃO DO GRAU DE LICENCIADO EM
ENGENHARIA INFORMÁTICA

Rafael Marques
Dezembro / 2023

POLITÉCNICO GUARDA

Escola Superior de Tecnologia e Gestão

Airsense
Qualidade do Ar Interior

PROJETO DE FIM DE CURSO
PARA OBTENÇÃO DO GRAU DE LICENCIADO EM
ENGENHARIA INFORMÁTICA

Professor Orientador: Celestino Gonçalves

Rafael Marques

Dezembro / 2023

Ficha de Identificação

Aluno:

Nome: Rafael Mendes Marques

Número: 1011220

Licenciatura: Engenharia Informática

Estabelecimento de Ensino:

Instituto Politécnico da Guarda (IPG)

Escola Superior de Tecnologia e Gestão (ESTG)

Docente Orientador de Estágio:

Nome: Celestino Gonçalves

Docente Co-orientador de Estágio:

Nome: Rui Pitarma

Resumo

O seguinte relatório descreve o projeto realizado no âmbito da unidade curricular Projeto de Informática, integrada na Licenciatura em Engenharia Informática da ESTG do Instituto Politécnico da Guarda.

O projeto teve como objetivo a criação de um sistema de monitorização da qualidade do ar num espaço interior. Com o uso de uma variedade de sensores ligados a um microcontrolador ESP32, o sistema mede certos parâmetros, como dióxido de carbono, compostos orgânicos voláteis, partículas, temperatura, humidade, e pressão. As medidas registadas pelos sensores são enviadas para uma base de dados InfluxDB através de Wi-Fi, onde são acedidas pelo software Grafana, que gera uma interface gráfica onde se pode visualizar um histórico de todos os valores recolhidos. Os valores podem ser visualizados em vários estilos gráficos consoante a preferência do utilizador, como tabelas, gráficos temporais, histogramas, e outras opções.

Inicialmente foi feito o estudo de vários tipos de microcontroladores e sensores para conseguir selecionar as melhores opções consoante fatores como o custo, tamanho, e precisão das medidas.

Juntamente com o protótipo final, foi também desenhada uma placa de circuito impressa, e uma caixa para impressão 3D, que facilitaram a junção dos vários componentes do hardware.

Por fim, foi feito um teste de duas semanas com o hardware final, em situação real, em ambiente escolar, que serviu para validar os resultados e o funcionamento do sistema.

Palavras-chave: Microcontrolador, InfluxDB, Grafana, ESP32, Qualidade do Ar Interior.

Abstract

The following report describes the project carried out within the scope of the academic course "Computer Project", integrated into the bachelor's program in Computer Engineering at ESTG, Polytechnic Institute of Guarda.

The project aimed to create an indoor air quality monitoring system. By using a variety of sensors connected to an ESP32 microcontroller, the system measures specific parameters such as carbon dioxide, volatile organic compounds, particulate matter, temperature, humidity, and pressure. The sensor-recorded measurements are sent to an InfluxDB database via Wi-Fi, where they are accessed by Grafana, which generates a graphical user interface that displays a history of all collected values. The values can be visualized in various graphical styles according to the user's preference, including tables, time-series charts, histograms, and other options.

Initially, some research was conducted to evaluate various types of microcontrollers and sensors to select the best options based on factors such as cost, size, and measurement accuracy.

A printed circuit board and a 3D-printed enclosure were designed alongside the final prototype, which facilitated the integration of the various hardware components.

Lastly, a two week-long test was conducted using the final hardware, in order to validate the overall results and operating behavior of the system.

Keywords: Microcontroller, InfluxDB, Grafana, ESP32, Indoor Air Quality.

Índice

Ficha de Identificação iii

Resumo iv

Abstract v

Índice de Figuras viii

Índice de Tabelas x

Lista de siglas e acrónimos xi

1. Introdução 1

1.1. Enquadramento e Motivação 1

1.2. Descrição do Problema 1

1.3. Objetivos 2

1.4. Estrutura do Relatório 2

2. Estado da Arte 4

2.1. Qualidade do Ar Interior 4

2.1.1. Poluentes e os Seus Efeitos na Saúde Humana 4

2.2. Trabalho Relacionado 6

2.3. Soluções Comerciais Existentes 9

3. Definição de Requisitos 12

3.1. Requisitos Funcionais 12

3.2. Requisitos não Funcionais 12

3.3. Tecnologias 13

4. Arquitetura do Hardware 15

4.1. Microcontrolador 15

4.2. Sensores 16

4.3. Outros Dispositivos 17

4.4. PCB 19

4.5. Caixa 22
4.6. Protótipos Exploratórios 25
4.6.1. Protótipo 1 25
4.6.2. Protótipo 2 26
4.6.3. Protótipo 3 27
4.6.4. Protótipo 4 28
4.7. Protótipo Final 31
5. Software do Sistema 33
5.1. Funções 33
5.2. Wi-Fi Manager 43
5.3. InfluxDB 45
5.4. Grafana 46
6. Verificação e Testes 50
6.1. Análise de Resultados 50
7. Conclusão 58
7.1. Trabalho Futuro 58
Referências 60
Anexos 62
A1 - Ficheiro main.cpp 62
A2 - Ficheiro Platformio.ini 72
A3 - Diagrama de pinout ESP32 DevKit 74
A4 - Flux Queries 75

Índice de Figuras

Figura 2.1 - Temtop M200C 10
Figura 2.2 - SAF Aranet4 11
Figura 4.1 - Arquitetura do hardware 15
Figura 4.2 - Interface gráfica do ecrã 18
Figura 4.3 - Rotary encoder, buzzer, e leds do sistema 18
Figura 4.4 - Diagrama esquemático da PCB 19
Figura 4.5 - Layout da PCB gerado automáticamente pelo EasyEDA 20
Figura 4.6 - Design final da PCB 20
Figura 4.7 - Design final da PCB 21
Figura 4.8 - PCB final com componentes soldados 21
Figura 4.9 - Caixa, revisão 1 22
Figura 4.10 - Caixa, revisão 2 23
Figura 4.11 - Caixa, revisão 3 23
Figura 4.12 - Caixa, design final 24
Figura 4.13 - Suporte vertical 24
Figura 4.14 - Primeiro protótipo 25
Figura 4.15 - Segundo protótipo 26
Figura 4.16 - Terceiro protótipo 28
Figura 4.17 - Quarto protótipo 29
Figura 4.18 - Layout dos componentes do protótipo final 30
Figura 4.19 - Hardware final 31
Figura 5.1 - Arquitetura do software 33
Figura 5.2 - Ecrã inicial do sistema 34
Figura 5.3 - Interface gráfica do ecrã 39
Figura 5.4 - Funcionalidade de alteração da taxa de atualização 43
Figura 5.5 - Seleção do access point em Android 44
Figura 5.6 - Seleção de rede no WiFiManager 45
Figura 5.7 - Interface da InfluxDB 46
Figura 5.8 - Gráficos de temperatura, humidade, pressão, co2, e VOC Index 48
Figura 5.9 - Gráfico temporal do nível de partículas 48
Figura 5.10 - Valores máximos e mínimos diários 49

Figura 5.11 - Layout mobile 49
Figura 5.12 - Layout mobile 2 49
Figura 6.1 - Segunda-feira dia seis de novembro 51
Figura 6.2 - Horário da sala 39, ESTG 51
Figura 6.3 - Terça-feira dia sete do novembro 52
Figura 6.4 - Quinta-feira dia nove de novembro 53
Figura 6.5 - Sexta-feira dia 10 de novembro 53
Figura 6.6 - Segunda-feira dia 13 de novembro 54
Figura 6.7 - Quarta-feira dia 15 de novembro 54
Figura 6.8 - Temperatura, dia 10-14 novembro 55
Figura 6.9 - Teste com desumidificador 56
Figura 6.10 - Teste do sensor de VOC 56
Figura 6.11 - Teste do sensor de partículas 57

Índice de Tabelas

Tabela 2.1 - Poluentes e efeitos na saúde 5
Tabela 2.2 - Comparação entre projetos existentes 9
Tabela 4.1 - Especificações técnicas dos sensores utilizados 16
Tabela 4.2 - Hardware do primeiro protótipo 26
Tabela 4.3 - Hardware do segundo protótipo 27
Tabela 4.4 - Hardware do terceiro protótipo 28
Tabela 4.5 - Hardware do quarto protótipo 30
Tabela 4.6 - Componentes finais 32

Lista de siglas e acrónimos

AQI	*Air Quality Index*
CAD	*Computer-Aided Design*
CAM	*Computer-Aided Manufacture*
CO	*Carbon Monoxide*
CO2	*Carbon Dioxide*
EPA	*Environmental Protection Agency*
I2C	*Inter-Integrated Circuit*
IAQ	*Indoor Air Quality*
IDE	*Integrated development environment*
IoT	*Internet of Things*
LCD	*Liquid Crystal Display*
LED	*Light-emitting diode*
MCU	*Micro-controller Unit*
NH3	*Ammonia*
NO	*Nitric Oxide*
NO2	*Nitrogen Dioxide*
OLED	*Organic light-emitting diode*
PCB	*Printed Circuit Board*
PLA	*Polylactic acid*
PM	*Particulate Matter*
SBC	*Single-board computer*
SBS	*Sick Building Syndrome*
SQL	*Structured query language*
TFT	*Thin-film-transistor*
TVOC	*Total Volatile Organic Compounds*
VOC	*Volatile Organic Compounds*

1. Introdução

O seguinte relatório descreve o projeto AirSense, um sistema de monitorização da qualidade do ar interior desenvolvido pelo aluno Rafael Mendes Marques no âmbito da Unidade Curricular Projeto de Informática, na Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão do Instituto Politécnico da Guarda.

1.1. Enquadramento e Motivação

O projeto teve como objetivo criar um sistema de monitorização de qualidade do ar interior para medição de dióxido de carbono, partículas, temperatura, humidade relativa, pressão, e compostos orgânicos voláteis. Um dos pontos importantes para este projeto é o custo total do sistema. Dado o preço elevado de alternativas comerciais, é importante criar um sistema de baixo custo com capacidade de competir em certos aspetos com essas alternativas mais caras.

Especialmente depois da era pós COVID-19, tornou-se cada vez mais importante ter em consideração a qualidade do ar que as pessoas respiram. A poluição do ar tem um impacto direto na qualidade de vida das pessoas, certos aspetos como por exemplo uma concentração elevada de partículas no ar podem aumentar o risco de transmissão de vírus e bactérias através das vias respiratórias. Outro dos fatores importantes para a qualidade do ar interior é a concentração de dióxido de carbono, níveis de concentração de dióxido de carbono acima do recomendado podem causar problemas como, dores de cabeça, problemas de respiração, cansaço elevado, irritação nos olhos, entre outros. Ao medir estes e outros fatores relacionados à qualidade do ar, é possível tentar reduzir os efeitos secundários referidos anteriormente.

1.2. Descrição do Problema

Uma má qualidade do ar interior pode ter vários efeitos adversos à saúde. Existem vários sistemas comerciais de monitorização da qualidade do ar, no entanto, a grande maioria destes sistemas tem um preço elevado, e algumas das funcionalidades são limitadas. Este projeto foi criado com a ideia de fornecer uma alternativa de baixo custo,

e ao mesmo tempo tentar melhorar certos aspetos e funcionalidades disponíveis nos sistemas mais caros.

1.3. Objetivos

Pretende-se com este trabalho implementar um sistema que consiga, com o uso de um microcontrolador, medir, e registar dados relativos à qualidade do ar. Os dados adquiridos são enviados para uma base de dados, onde são acedidos por um *software* que desenha gráficos temporais dos valores, para visualização e acesso simplificado.

Os principais objetivos para implementação do projeto são:

- Estudo de vários microcontroladores e sensores de modo a conseguir selecionar a melhor opção para o projeto. Os vários sensores são avaliados consoante fatores como, custo, precisão de medidas, e tamanho. O microcontrolador é avaliado principalmente pelas suas características e pelo custo.
- Montagem e calibração do sistema embebido com o microcontrolador, sensores, e o resto do *hardware* selecionado.
- Medição de dióxido de carbono, *VOC, PM2.5*, temperatura, humidade, e pressão.
- Envio dos dados recolhidos para uma base de dados, neste caso é utilizada a InfluxDB
- Implementação do software Grafana para gerar gráficos com a informação lida da base de dados. Os gráficos poderão ser acedidos online numa *GUI*.
- Criação de uma *PCB* e uma caixa 3D à medida para o sistema.

1.4. Estrutura do Relatório

Para além da Introdução, este documento tem mais sete capítulos:

O segundo capítulo contém algumas descrições de produtos semelhantes disponíveis no mercado, assim como uma comparação com o produto desenvolvido pelo aluno. No terceiro capítulo são descritos os requisitos funcionais e não funcionais do projeto, assim como as tecnologias utilizadas. O quarto capítulo descreve a arquitetura do *hardware*, o microcontrolador, sensores, outros dispositivos, assim como o processo de desenvolvimento da *PCB* e da caixa 3D. O quinto capítulo descreve o software do

sistema, é feita a análise ao código e é explicado como o mesmo interage com os diferentes elementos do *hardware*. No sexto capítulo é feita a verificação e validação dos dados obtidos no teste feito ao sistema. Por fim, no capítulo sete é apresentada a conclusão, assim como algumas ideias para melhorar o sistema no futuro.

2. Estado da Arte

Este capítulo descreve a pesquiza realizada sobre vários elementos relacionados com o projeto, como o funcionamento de sistemas de medição de qualidade do ar, opções comerciais alternativas, e comparações entre vários tipos de sensores e microcontroladores utilizados no projeto. É também feito estudo dos vários fatores que comprometem a qualidade do ar, incluindo tópicos como, dióxido de carbono, partículas e químicos no ar, e o significado de um bom ou mau nível em termos de qualidade do ar interior.

2.1. Qualidade do Ar Interior

Segundo a EPA, a qualidade do ar interior tem um impacto significante na saúde devido principalmente às seguintes razões [1]:

- As pessoas passam, em média, 90 porcento do seu tempo em espaços interiores, onde a concentração de alguns poluentes pode ser entre duas a cinco vezes maior do que a concentração em espaços exteriores.

-Pessoas que são mais suscetíveis a efeitos negativos de poluição do ar (crianças, idosos, pessoas com problemas cardiovasculares ou respiratórios) costumam passar ainda mais tempo dentro de casa.

- A concentração de certos poluentes em espaços interiores tem vindo a aumentar nas últimas décadas devido a fatores como o aumento do uso de produtos de beleza, pesticidas, produtos de limpeza, e o uso de produtos sintéticos de construção.

2.1.1. Poluentes e os Seus Efeitos na Saúde Humana

Na Tabela 2.1 são demonstrados os efeitos na saúde humana de alguns poluentes relativos à qualidade do ar interior [2] [3] [4].

Tabela 2.1 - Poluentes e efeitos na saúde

Poluente	**Efeitos na saúde humana**
VOCs (*Voltile Organic Compounds*)	- Irritação nos olhos, nariz, e garganta - Dores de cabeça, perda de coordenação e concentração -Dano ao fígado, rins, e sistema nervoso central - Cancro
CO (Monóxido de carbono)	- Fatiga - Dores peitorais - Problemas de visão
CO2 (Dióxido de carbono)	- Dores de cabeça - Cansaço - Irritação nos olhos e garganta -Perda de concentração
PM (*Particulate matter*, PM1.0, PM2.5, PM10.0)	- Irritação nos olhos, nariz, e garganta -Agravamento dos sintomas de doenças coronárias e respiratórias -Morte prematura (pessoas com doenças pulmonares ou cardíacas)
NO2 (Dióxido de nitrogénio)	- Asma - Problemas respiratórios
Radon	- Cancro pulmonar
Amianto	- Cancro pulmonar
Tabaco	- Cancro pulmonar - Problemas respiratórios - Doenças cardiovasculares

Embora efeitos adversos à saúde tenham sido atribuídos a alguns destes poluentes, a compreensão científica de alguns problemas em relação à qualidade do ar interior continua a evoluir. Um exemplo é o "Sick Building Syndrome", ou SBS, que ocorre quando ocupantes de um edifício desenvolvem sintomas semelhantes após entrarem num edifício específico, com sintomas que vão diminuindo após saírem do edifício.

Alguns Investigadores têm também estado a estudar a relação entre a qualidade do ar interior e certos problemas que não são tradicionalmente ligados à saúde, como o desempenho de estudantes, ou o nível de produtividade em ambientes de trabalho [1].

Outra área de investigação que está a evoluir é o design, construção operação e manutenção de edifícios "verdes" que alcançam eficiência energética e melhoram a qualidade do ar interior.

2.2. Trabalho Relacionado

De seguida, vão descrever-se alguns exemplos de projetos com características semelhantes.

AirVA [5]**:**

Este projeto consiste de um sistema de monitorização da qualidade do ar com a habilidade de detetar quantas pessoas estão presentes numa divisão. O sistema desenvolvido usa uma série de sensores ligados a vários microcontroladores, esses microcontroladores são depois ligados a um *SBC* Raspberry Pi através de Wi-Fi. O sistema usa os dados recolhidos pelos sensores, para calcular um valor de *index* relativo à qualidade do ar interior.

Devido ao preço elevado de todos os componentes usados no AirVa, é uma opção cara em relação ao Airsense, no entanto, o AirVa consegue detetar o número de pessoas na divisão, o que é uma grande vantagem e uma funcionalidade excelente para este tipo de sistemas.

iAQ+ [6]**:**

O iAQ+ é um sistema de baixo custo, que faz a monitorização de vários fatores para calcular um index relativo à qualidade do ar. Com o uso de um sensor BME680, o sistema consegue medir valores de temperatura, humidade, pressão, e VOC. O projeto incorpora também uma aplicação para smartphone e uma web app que permite fazer a visualização dos valores registados pelo sistema, assim como um gráfico histórico dos mesmos.

O sistema tem algumas semelhanças ao Airsense, nomeadamente o uso de um buzzer para gerar alertas sonoros, e o uso de LEDs para demonstrar o nível da qualidade do ar

de uma maneira visual. No entanto, o sistema não tem sensores de CO2, ou de partículas, que são fatores importantes para projetos nesta área.

BEVO [7]**:**

O projeto BEVO consiste no estudo e desenvolvimento se um sistema de monitorização de qualidade do ar com o nome de BEVO Beacon. O projeto focou-se num estudo intensivo dos valores recolhidos por 20 sistemas com o mesmo hardware. Estes sistemas recolhem valores de CO2, partículas, CO, NO2, e TVOC. De maneira semelhante ao Airsense, o BEVO Beacon usa uma PCB para ligar todos os componentes, e uma caixa à medida para o sistema, neste caso feita em contraplacado.

CampusEMonitor [8]**:**

O CampusEMonitor é um sistema de monitorização ambiente de baixo custo. Este sistema foi desenvolvido para monitorizar o ambiente em salas com diverso equipamento de redes como servidões e *switches*. Devido a algumas falhas de ar condicionado e ao calor gerado pelo equipamento, estas salas podem aquecer rapidamente, atingindo um nível tão elevado que os equipamentos se desligam automaticamente como proteção, causando falhas na rede. Noutros casos vários gases podem contaminar o ambiente e causar uma qualidade do ar não aceitável. Para tentar detetar estes aumentos de temperatura e gases, foi criado este sistema que usa um *SBC* Raspberry Pi, ligado a um sensor de temperatura e humidade DHT11, e um sensor de monóxido de carbono MQ-7. Os valores lidos pelos sensores são enviados para uma base de dados para serem usados numa *dashboard* gráfica.

Em relação ao Airsense, o CampusEMonitor é bastante limitado em termos de sensores, apenas medindo níveis de temperatura, humidade, e monóxido de carbono. Um erro do projeto CampusEMonitor é o uso do sensor MQ-7 para monitorização do CO2, visto que o sensor mede apenas CO.

IndoAirSense [9]**:**

Este projeto focou-se principalmente na análise e estudo extensivo dos dados recolhidos. Foi também desenvolvido um sistema para ler esses dados, que é capaz de fornecer previsões de curto prazo analisando os dados recolhidos. O sistema desenvolvido mede o nível de partículas, NO2, CO2, CO, temperatura, e humidade. Foi

também criada uma aplicação móvel para android onde são demonstrados os gráficos com todos os dados recolhidos.

Design and Simulation of Environment Indoor Air Quality Monitoring and Controlling System using IoT Technology [10]**:**

Este projeto consistiu na criação de um sistema de monitorização da qualidade do ar. O sistema usa um SBC Arduino Mega, ligado a 6 sensores, que medem os níveis de CO, CO2, NO2, O3, PM, VOC, temperatura, e humidade. Os dados são enviados por Wi-Fi para uma aplicação móvel. De forma semelhante ao Airsense, este sistema implementou vários dispositivos além de sensores, como um ecrã para visualização dos dados, um *buzzer* e um LED, e adicionalmente, ventoinhas para o controlo do fluxo de ar.

PCFN (Personal Confort Feedback Node) [11]**:**

Este projeto apresenta uma plataforma de recolha, gestão, e armazenamento de medidas relativas à qualidade do ar. A plataforma consiste de um grupo de sistemas PCFN, cada um destes protótipos contem vários sensores ligados a um microcontrolador, assim como um ecrã sensível ao toque que permite que os utilizadores deem feedback acerca da qualidade do ar interior. Os sistemas PCFN medem os níveis de VOC, CO2, humidade e temperatura, os dados adquiridos pelos vários sistemas são enviados por Wi-Fi para uma base station, que guarda todos os dados numa base de dados PostgreSQL.

Na Tabela 2.2 é apresentada uma comparação de alguns fatores dos vários projetos analisados.

Tabela 2.2 - Comparação entre projetos existentes

Projeto	Airsense	AirVA	iAQ+	BEVO	Campus EMonitor	IndoAirSense	EIAQ System	PCFN
MCU ou SBC	ESP32	Raspberry Pi	ESP8266	Raspberry Pi	Raspberry Pi	Não referido	Arduino Mega	Raspberry Pi, Arduino
Parâmetros monitorizados	CO2, T, H, P, PM, VOC	CO2, T, H, PM, NH3, NO, Álcool, Benzeno, Fumo, Pó	T, H, P, VOC	CO2, PM, CO, NO2, TVOC	T, H, CO	PM, NO2, CO2, CO, T, H	CO, CO2, NO2, O3, VOC, PM, T, H	VOC, CO2, H, T
Conetividade	Wi-Fi	Wi-Fi / Zigbee	Wi-Fi	×	Wired Ethernet	Não referido	Wi-Fi	Wi-Fi
Notificações	×	√	√	×	√	Não referido	√	×
Acesso aos dados	Web	Mobile app, Web	Mobile app, Web	Ficheiro local	Web	Mobile app	Mobile app	Database
Open-Source	√	√	√	√	√	×	×	√
Low-Cost	√	×	√	×	√	√	√	×

2.3. Soluções Comerciais Existentes

Atualmente, existem centenas de produtos de monitorização da qualidade do ar no mercado. Para esta secção, foram escolhidos dois dos produtos mais populares, e será apresentado um breve sumário dos mesmos assim como comparações de funcionalidades com o AirSense.

O **M200C** (Figura 2.1) é das opções mais usadas da **Temtop**, um dos líderes de mercado em sensores de qualidade do ar. É um sistema compacto e portátil que faz medições de

dióxido de carbono, PM2.5, PM10, temperatura, e humidade. Tem uma interface gráfica onde é possível visualizar gráficos temporais com os valores de todos os sensores.

A maior vantagem deste sistema relativamente ao Airsense é o facto de ser uma solução portátil com uma bateria recarregável. No entanto, este sistema não oferece Wi-Fi, o que faz com que seja impossível a visualização dos dados online.

Figura 2.1 - Temtop M200C

Fonte: https://www.amazon.com/Temtop-Generation-Recording-Calibration-Temperature/dp/B0928Z1FBG/ref=sr_1_3

O **SAF Aranet4** (Figura 2.2), é a opção mais popular na Amazon de momento. O sistema faz medições de dióxido de carbono, temperatura, humidade, e pressão atmosférica. Tem a possibilidade de ser ligado por Bluetooth a um smartphone através de uma app, onde é possível ver gráficos temporais das medidas.

A maior vantagem deste sistema relativamente ao Airsense é o facto de ser uma solução portátil com uma bateria recarregável. O facto de ter um ecrã de *e-ink* e não estar

constantemente ligado a uma rede faz com que o sistema tenha uma bateria que dura até 4 anos, o que torna o produto muito mais apelativo e prático. Uma desvantagem comparativamente com o Airsense é a falta de um sensor de medição de partículas.

Figura 2.2 - SAF Aranet4

Fonte: https://www.amazon.com/Aranet4-Home-Temperature-Ink-Configuration/dp/B07YY7BH2W/ref=sr_1_3

3. Definição de Requisitos

Neste capítulo são apresentados os requisitos que o sistema deve cumprir, dividido em duas secções, requisitos funcionais, e requisitos não funcionais.

O objetivo principal do projeto é criar um sistema que consiga ler dados de vários sensores com o uso de um microcontrolador. Esses dados devem ser enviados para uma base de dados *online*, onde podem são acedidos por um *software* que irá gerar gráficos com toda a informação recolhida. Pretende-se ter um sistema de baixo custo, fácil de usar, com uma arquitetura aberta, e uma interface gráfica simples.

3.1. Requisitos Funcionais

De seguida apresentam-se os requisitos funcionais estabelecidos:

- Medições – O sistema deve efetuar medidas de certos parâmetros relativos à qualidade do ar interior usando os sensores disponíveis;
- Ligação à internet – O sistema deve ser capaz de fazer a ligação a uma rede por Wi-Fi;
- Armazenamento de dados- O sistema deve conseguir enviar as medidas efetuadas para uma base de dados online;
- Visualização de dados – Deve ser criada uma *dashboard* onde se possam visualizar as medidas com uma interface gráfica;
- Seleção da taxa de leitura dos sensores – O utilizador deve conseguir alterar facilmente a taxa de atualização dos sensores e do *upload* dos valores lidos.

3.2. Requisitos não Funcionais

De seguida apresentam-se os requisitos não funcionais:

- Usabilidade – O sistema deve ser fácil de usar e a interface gráfica deve ser intuitiva para o utilizador;

- Escalabilidade – O sistema deve ser escalável para poder acomodar mais sensores em futuras versões;
- Estética – O sistema deve ter um *design* visualmente apelativo;
- Ergonomia – O sistema deve ser fácil de transportar e desmontar.

3.3. Tecnologias

Para cumprir os requisitos listados anteriormente, foram utilizadas as seguintes tecnologias.

O **ESP32** é um microcontrolador de baixo consumo de energia que integra Wi-Fi e Bluetooth. Oferece alta flexibilidade e versatilidade para projetos de IoT e sistemas embebidos [12].

O **Grafana** é uma plataforma de visualização de dados que é maioritariamente usada em projetos de monitorização e análise. Permite a criação de painéis interativos e gráficos personalizados. É um software extremamente flexível e escalável. Tem uma interface intuitiva que facilita o acompanhamento de dados em tempo real, o que faz com que seja uma ferramenta essencial para a gestão e otimização de sistemas, servidores, e aplicações, especialmente relacionados com IoT [13].

O **InfluxDB** é uma base de dados de séries temporais. É utilizada para o armazenamento de dados que evoluem ao longo do tempo. Graças à facilidade de integração com outras ferramentas, como o Grafana, tornou-se uma boa escolha para análise e visualização de dados temporais em diversos tipos de projetos.

O **Visual Studio Code** da Microsoft, é um IDE leve e altamente personalizável para desenvolvimento de *software*. É um IDE *open-source*, e com suporte para centenas de linguagens de programação. Tem uma interface amigável e suporta uma vasta gama de extensões que facilitam o desenvolvimento de projetos [14].

O **PlatformIO** é uma ferramenta multiplataforma e multi-arquitetura feita para o desenvolvimento de *software* em sistemas embebidos. Com a extensão para o Visual Studio Code, oferece a possibilidade de compilação e *upload* de código diretamente do IDE para placas Arduino, ESP32, e outros microcontroladores. Esta funcionalidade permite o uso de IDEs alternativos ao ArduinoIDE, que é extremamente limitado.

O **ArduinoIDE** é um IDE de código aberto, é utilizado para programar placas Arduino. Oferece uma interface simples a acessível para programadores, no entanto, comparado com outros IDEs, é bastante limitado, e não oferece muitas das funcionalidades que atualmente são consideradas indispensáveis, como por exemplo ferramentas robustas para debugging, ou controlo de versões.

A linguagem **C++** é uma linguagem de programação de alto nível que combina aspetos da linguagem C com programação orientada a objetos. C++ é uma escolha comum para projetos que requerem alto desempenho.

O **Fusion360** é um software de modelação 3D e CAD/CAM da Autodesk. É um *software* muito usado nas áreas de engenharia, design industrial, e fabrico, pois proporciona ferramentas que abrangem todo o ciclo de vida de um produto, desde o conceito inicial, até à produção [15].

O **EasyEDA** é uma plataforma de design eletrónico que permite a criação de esquemas, design de PCBs, e simulações de circuitos. Tem uma vasta biblioteca de componentes o que faz com que seja uma opção acessível e eficiente para projetos de eletrónica com vários níveis de complexidade [16].

4. Arquitetura do Hardware

Neste capítulo será apresentada a implementação do projeto final, incluindo as tecnologias utilizadas, a arquitetura do hardware, e o processo de *design* de certos componentes.

A Figura 4.1 representa a arquitetura do *hardware*. O sistema é composto por vários sensores, e outros elementos funcionais ligados ao microcontrolador através de uma PCB. Essa PCB é depois montada numa caixa impressa numa impressora 3D.

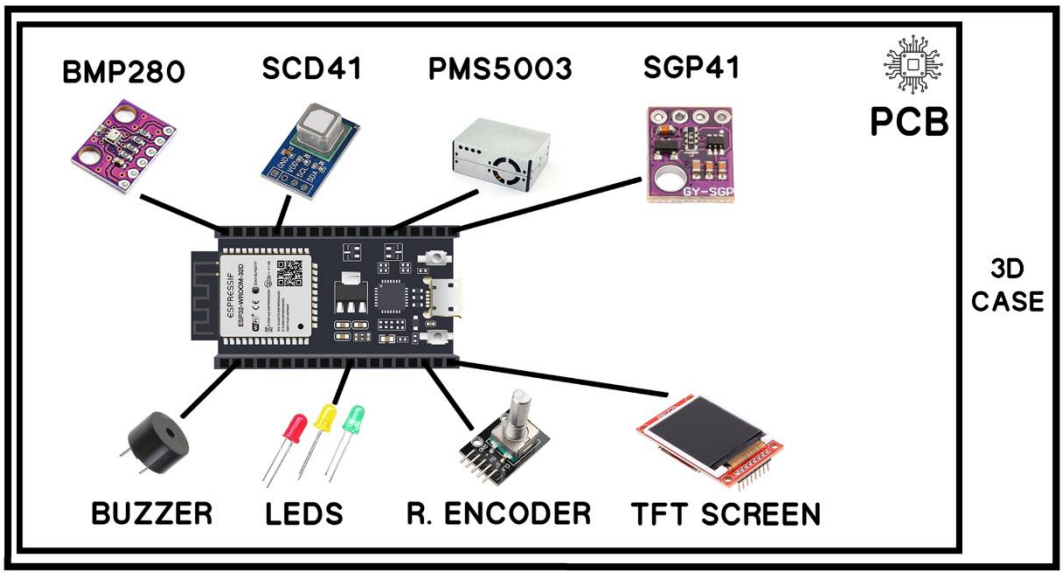


Figura 4.1 - Arquitetura do hardware

4.1. Microcontrolador

Depois dos vários testes descritos no capítulo anterior, o microcontrolador selecionado foi o ESP32.

O ESP32 foi escolhido devido ao seu preço competitivo e quantidade elevada de características desejadas para o projeto. Uma unidade custa apenas cerca de quatro euros, tem um processador rápido, Bluetooth e Wi-Fi integrados, e bastante memória. É também mais compacto do que outras alternativas da Arduino. Suporta vários estilos de interface como, SPI, I2C, UART, entre outros, o que faz com que seja ideal para um projeto de IoT com vários tipos de sensores como este.

4.2. Sensores

O sistema final usa quatro sensores para medir a Temperatura, Humidade relativa, Pressão barométrica, VOC, CO2, e Partículas. Na Tabela 4.1 são detalhados os sensores escolhidos juntamente com as suas características técnicas.

Tabela 4.1 - Especificações técnicas dos sensores utilizados

Modelo	BME280	PMS5003	SCD41	SGP40
Fabricante	Bosch	Plantower	Sensirion	Sensirion
Parâmetros monitorizados	Temperatura, Humidade, Pressão	PM1.0, PM2.5, PM10.0	CO2	VOC
Protocolo de comunicação	I2C e SPI	Serial e UART	I2C	I2C
Intervalo de medidas	P: 300…1100 hPa T: -40…+85°C H: 0…100%	0…1000µg/m³	400…5000ppm	0…500 VOC Index
Precisão	Temperatura: ±1.0°C Humidade: ±3% Pressão: ±1 hPa	±10%@100 a 500µg/m3	±50 ppm	±15% m.v.
Princípio de medição	Capacitativo, piezo-resistivo	Dispersão laser	Foto-acústico	MOx
Custo	2.5€	16€	21€	6.3€

Para medir a Temperatura, Humidade, e pressão, foi utilizado o sensor BME280, a comunicação com o microcontrolador é feita por protocolo I2C. Os detalhes mais específicos do sensor não são públicos visto ser tecnologia proprietária, mas a temperatura é medida através de mudanças em voltagem de um díodo, a humidade é medida através de um princípio capacitativo, e a medida de pressão é calculada usando um sensor piezo-resistivo [17].

O sensor de partículas escolhido foi o PMS5003. Este sensor mede o número de partículas no ar com um diâmetro de 1, 2.5, e 10 micrómetros. O sensor funciona através do princípio de dispersão de um laser. Com o uso de uma ventoinha de *intake* as partículas passam por um laser focado num ponto. As partículas causam com que a luz do laser

disperse, o que é detetado por um díodo sensível à luz. O valor registado por esse díodo é depois convertido para a concentração de partículas com ajuda do microprocessador no sensor [18].

Para sensor de Dióxido de Carbono, usou-se o Sensirion SCD41. O sensor comunica com o microcontrolador através do protocolo I2C, e usa uma tecnologia foto acústica para medição do nível de CO2, que é um método relativamente novo neste tipo de sensores. O sensor emite uma luz infravermelha, quando esta luz é absorvida pelas moléculas de CO2, elas começam a vibrar e a aquecer e arrefecer rapidamente. Devido a esta expansão e contração térmica, há mudanças de pressão e vibrações dentro do sensor, essas vibrações geram sinais acústicos que são medidos por um microfone no sensor. Com os sinais desse microfone, é possível fazer a conversão para um valor representativo da concentração de CO2 [19].

Para medição dos compostos orgânicos voláteis (VOC) foi escolhido o sensor Sensirion SGP40. O SGP40 deteta a concentração de VOCs no ar e regista um valor de 0 a 500, onde 0-100 representa uma qualidade do ar excelente, de 100-200 qualidade boa, de 200-300 ar ligeiramente poluído, 300-400 moderadamente poluído, e 400-500 ar extremamente poluído [20].

4.3. Outros Dispositivos

Além dos sensores, o sistema tem também alguns elementos para proporcionar certas características mais práticas ao utilizador.

- Ecrã TFT (Figura 4.2). Apesar do método de uso principal do sistema ser por ligação Wi-Fi com a transmissão de dados para a *cloud*, é útil ter um ecrã para demonstrar os dados localmente a nível de *hardware*;

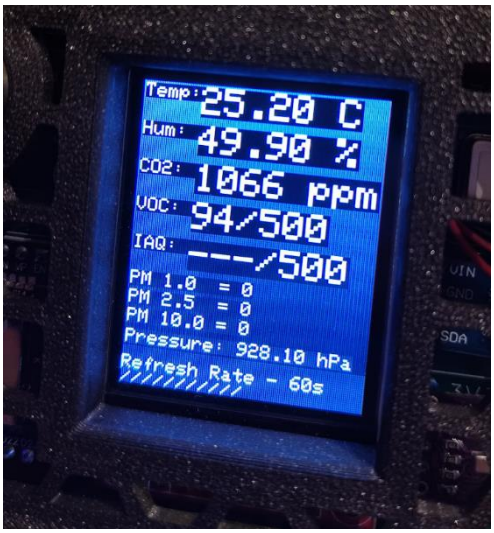


Figura 4.2 - Interface gráfica do ecrã

- Um *rotary encoder* que serve para alterar a taxa de atualização dos sensores e o intervalo de tempo em que os dados são enviados para a base de dados. Em certos casos, pode ser útil ter um intervalo mais longo, por exemplo de cinco em cinco minutos, para não ter excesso de *datapoints*. Noutros casos um intervalo mais curto pode ser preferível;
- Uma coluna de som que alerta para quando o sistema faz a ligação à rede e à base de dados;
- Um conjunto de 3 leds de cores diferentes que mostram visualmente o nível de qualidade do ar interior.

Na Figura 4.3 demonstra-se a implementação dos 3 elementos referidos anteriormente.

Figura 4.3 - Rotary encoder, buzzer, e leds do sistema

4.4. PCB

O desenvolvimento da PCB começa pela criação de um esquema com todos os componentes e as suas ligações. Como o EasyEDA tem uma biblioteca de componentes maioritariamente criada por utilizadores, muitos dos componentes necessários já estão disponíveis nessa biblioteca. Depois de fazer todas as ligações corretamente consoante o protótipo do sistema, ficamos com um diagrama que é demonstrado na Figura 4.4.

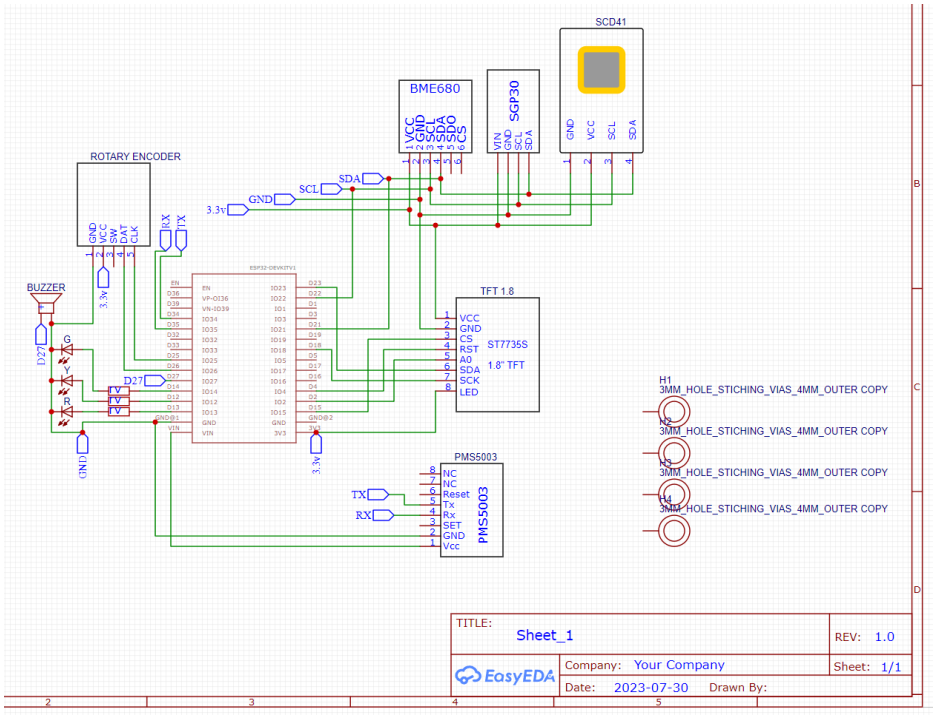


Figura 4.4 - Diagrama esquemático da PCB

Com este diagrama, é possível gerar uma PCB. O software, gera uma PCB com os componentes colocados aleatoriamente, que por seguinte têm de ser ajustados consoante as medidas desejadas. Depois de colocar os componentes nos sítios corretos e verificar todas as medidas, gera-se automaticamente o *wiring* para ficar com o design final.

Por fim, basta exportar o ficheiro *gerber*, e fazer *upload* para o fabricante escolhido, neste caso, a PCB foi fabricada pela JLCPCB.

Nas figuras seguintes, é demonstrado o processo de design final da PCB. A Figura 4.5 contém todos os componentes da PCB, assim como as ligações entre eles.

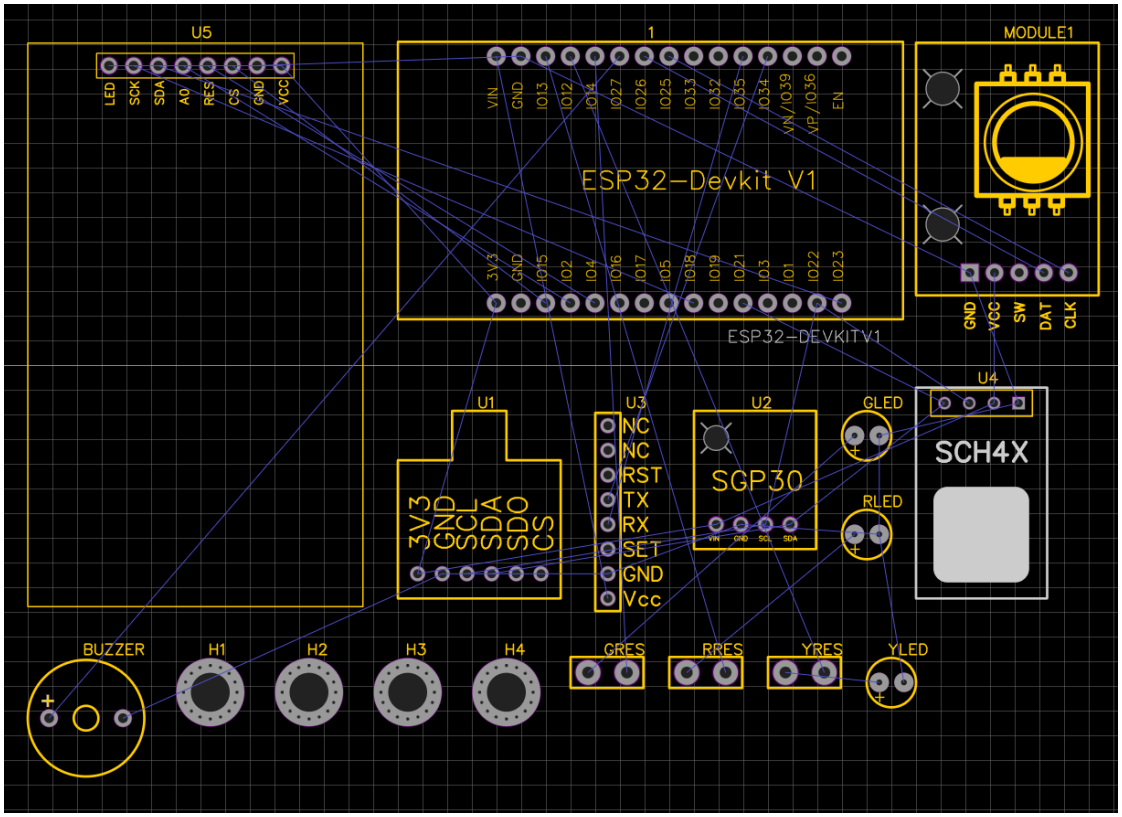


Figura 4.5 - Layout da PCB gerado automáticamente pelo EasyEDA

Na Figura 4.6, demonstra-se o layout final da PCB em modo *wireframe*, e em modo 3D na Figura 4.7.

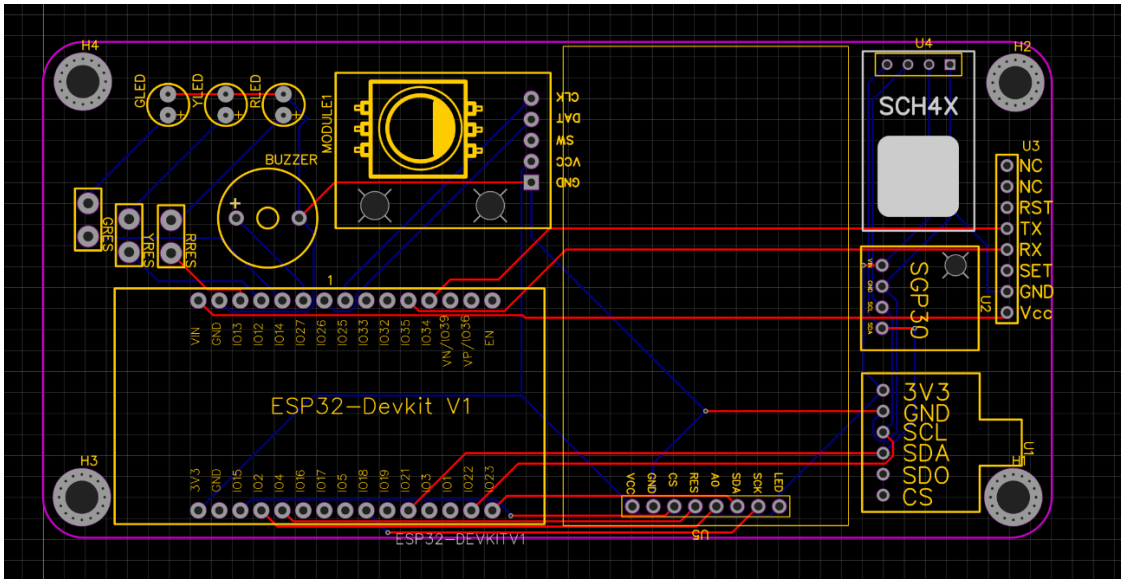


Figura 4.6 - Design final da PCB

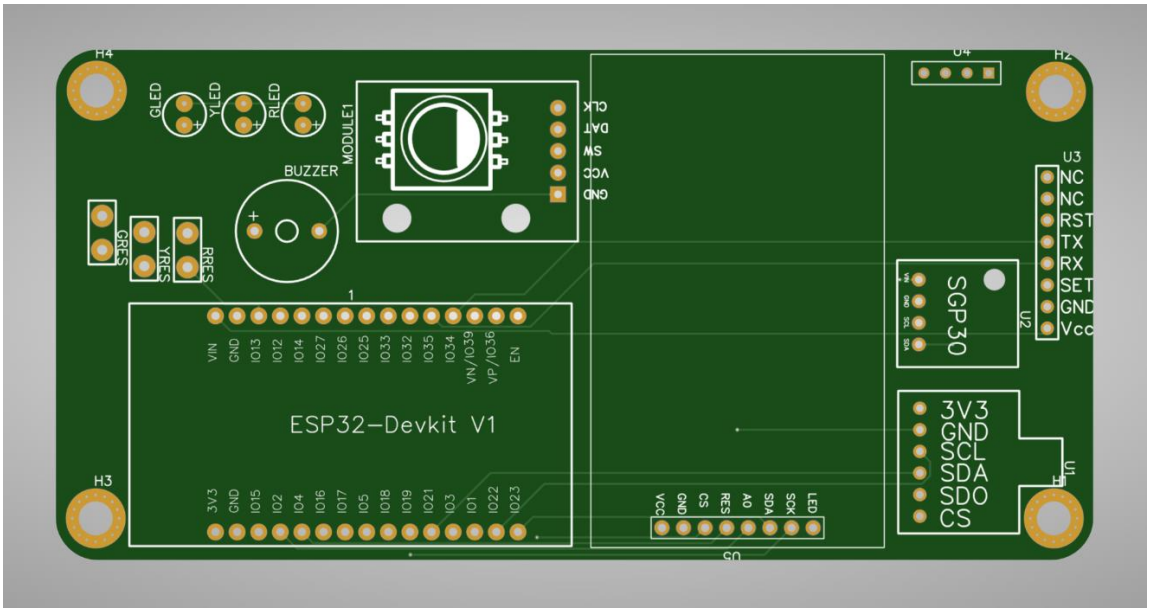


Figura 4.7 - Design final da PCB

Por fim, a Figura 4.8 mostra o layout final do sistema com todos os componentes soldados à PCB física.

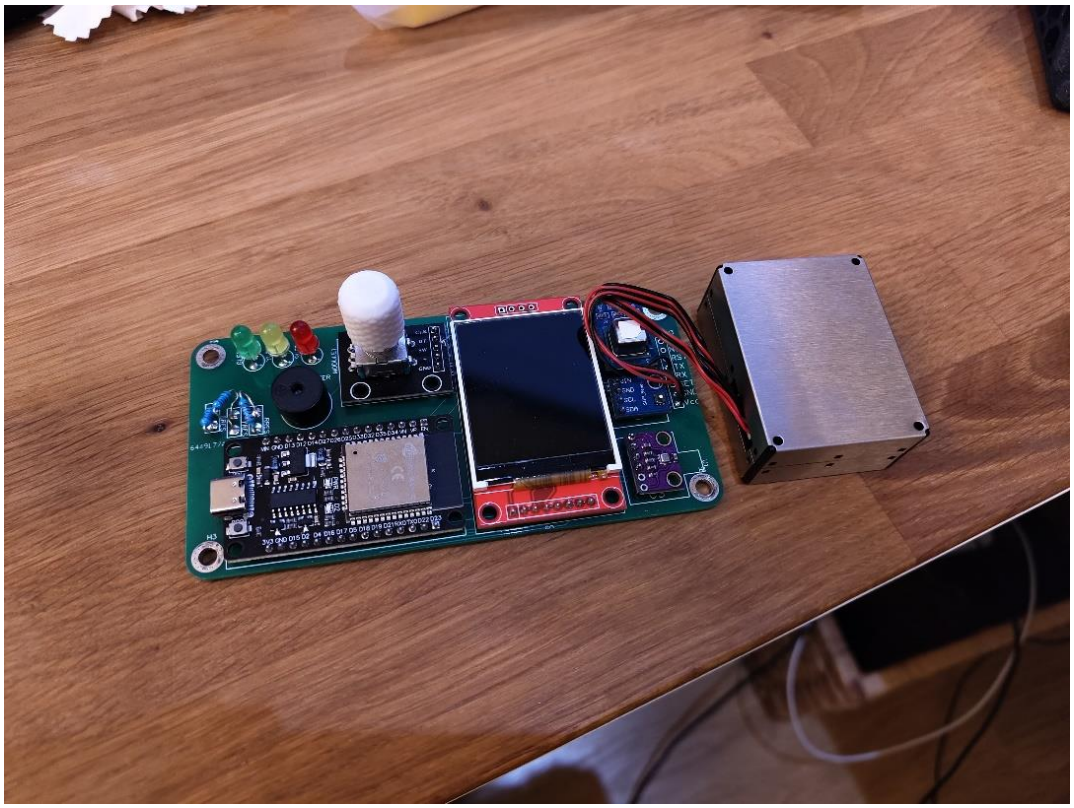


Figura 4.8 - PCB final com componentes soldados

4.5. Caixa

Para a caixa começa-se por criar o modelo 3D no Fusion360. O objetivo é criar um modelo simples, compacto, e que seja fácil de imprimir numa impressora 3D.

O modelo é gerado com formas simples, começando pelo *layout* plano com os *standoffs* para os parafusos, e algo para segurar o sensor de partículas, que é o único componente não fixo à PCB. Nas seguintes figuras, são demonstradas as diferentes etapas do design, desde o *layout* inicial básico, até à caixa completa. Na Figura 4.9 foi criada uma base simples, com os *standoffs* para os parafusos, e uma borda para segurar o sensor de partículas.

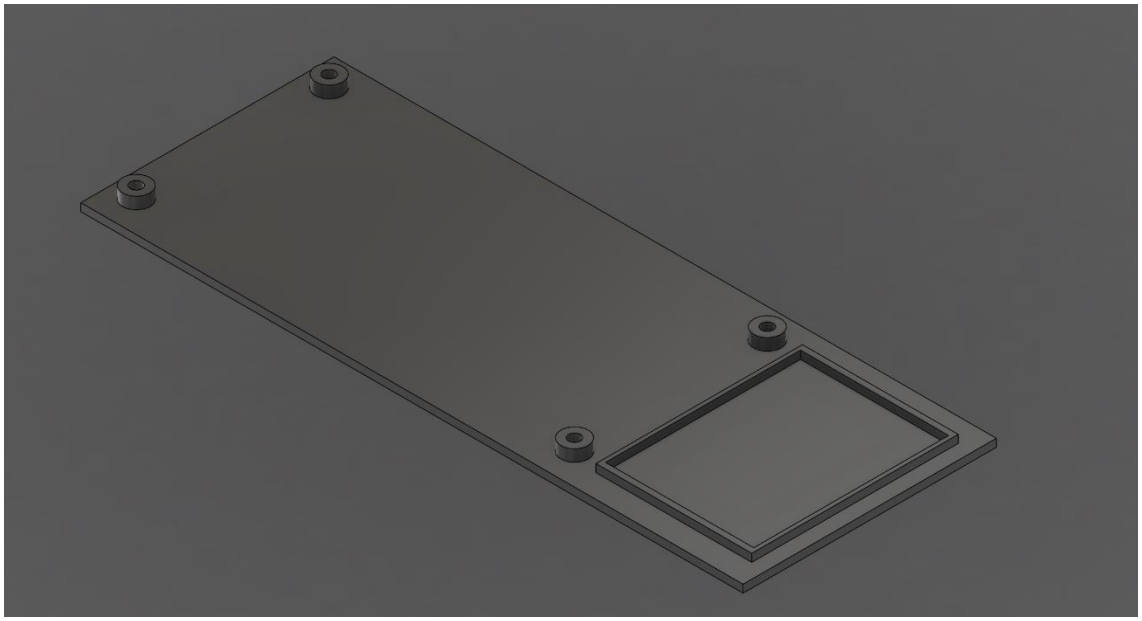


Figura 4.9 - Caixa, revisão 1

A Figura 4.10 demonstra o próximo passo, em que foi criada uma borda à volta da base inicial, assim como espaço para colocar os vários imans, e a ventilação para o sensor de partículas

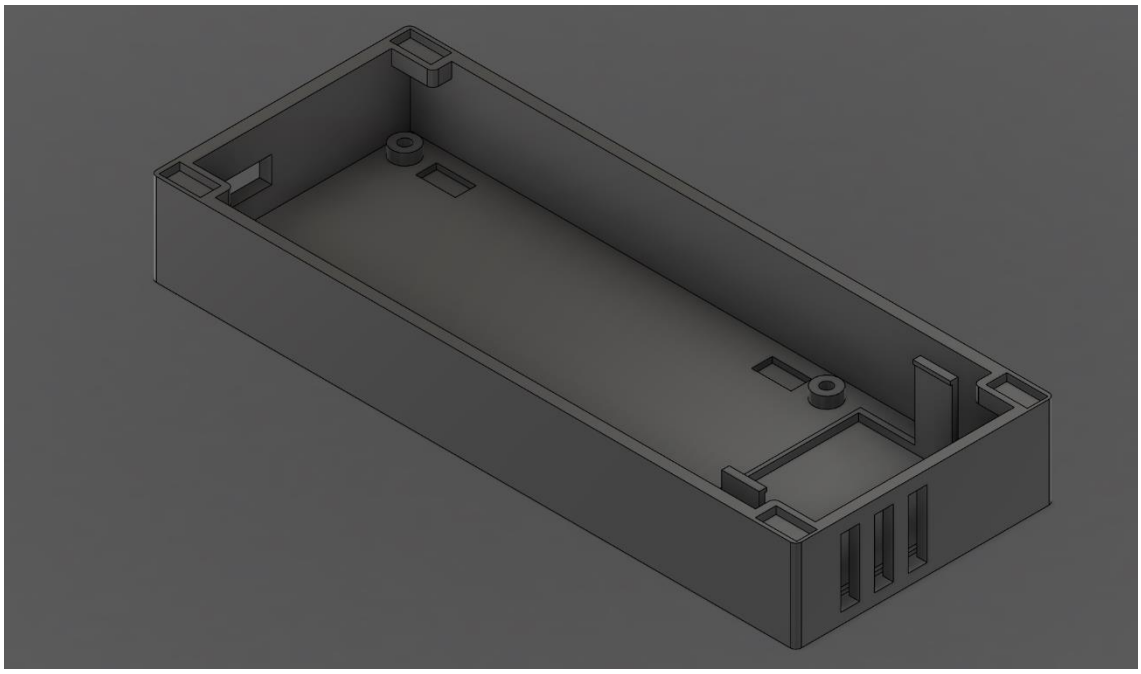


Figura 4.10 - Caixa, revisão 2

A Figura 4.11 demonstra o segundo elemento da caixa, que monta à primeira peça com imans. Este segundo elemento tem várias cortes para os outros elementos do sistema, os leds, o *rotary encoder*, o ecrã, e ventilação para os sensores.

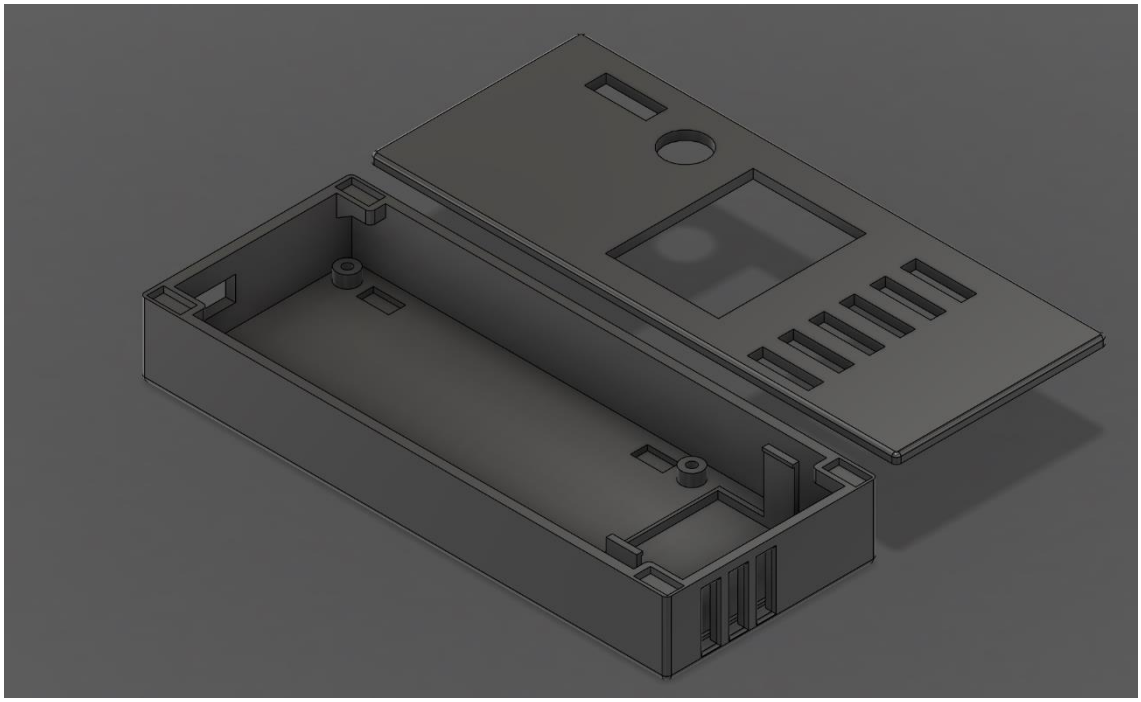


Figura 4.11 - Caixa, revisão 3

Na Figura 4.12 é demonstrado o design final com alguns ajustes, incluindo alguns cortes nas bordas para que estas não ficassem tão afiadas, e foi também adicionado um padrão de buracos em hexágono para aumentar o nível de ventilação do sistema.

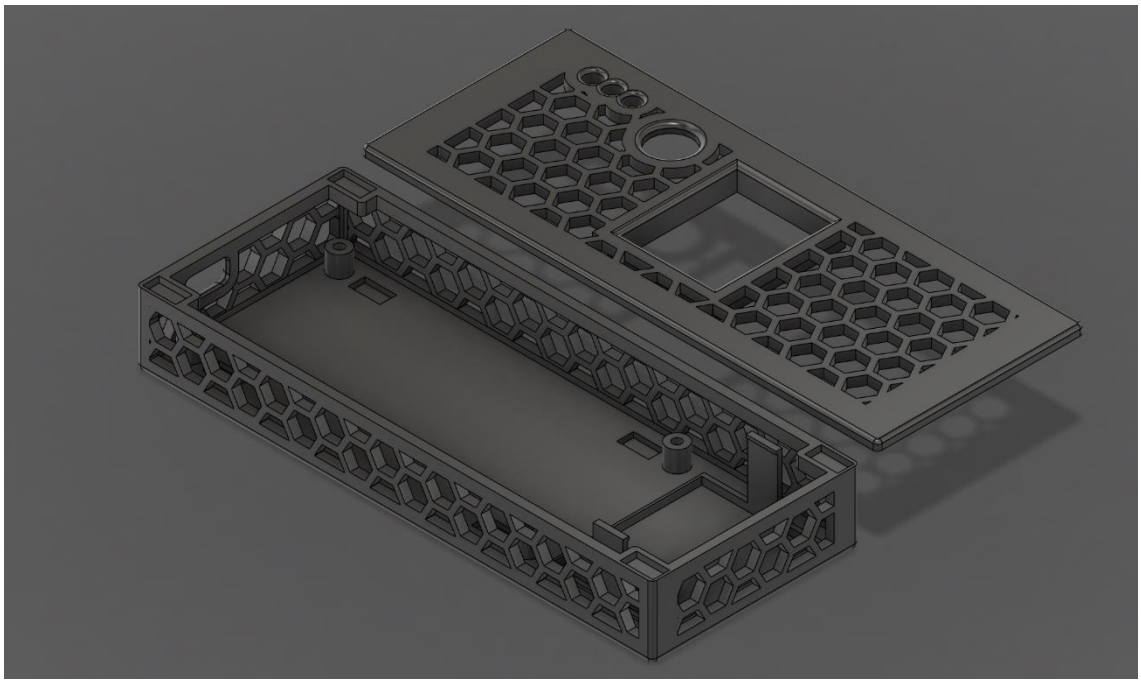


Figura 4.12 - Caixa, design final

Por fim, foi feito o suporte da Figura 4.13 para orientar o sistema na vertical.

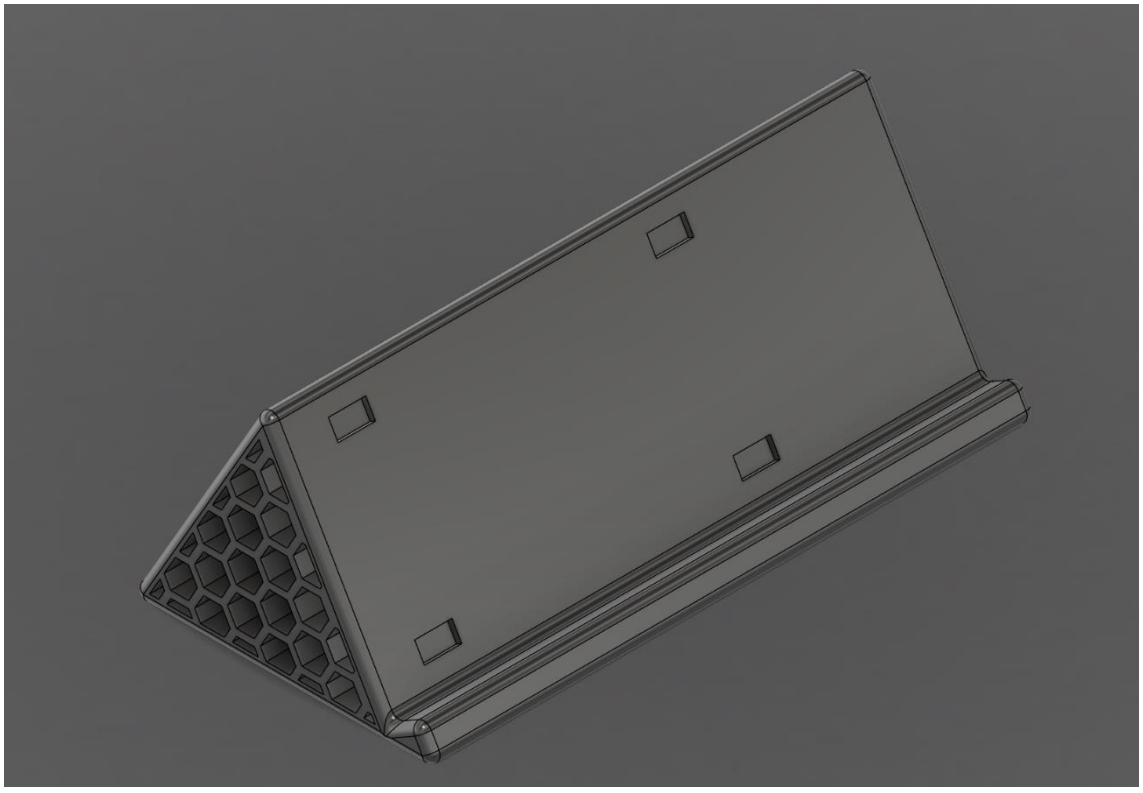


Figura 4.13 - Suporte vertical

Para juntar as três peças, são usados ímans colados em cada um dos componentes.

Com o modelo 3D feito, resta a impressão. Os componentes foram impressos numa impressora 3D Bambu Lab P1P em PLA, que é um plástico fácil de imprimir, barato, e resistente.

4.6. Protótipos Exploratórios

Ao longo do processo de criação do projeto, foram idealizados vários protótipos exploratórios. Estes protótipos tiveram como objetivo testar a qualidade de vários microcontroladores e sensores, e escolher qual o hardware mais apropriado para o sistema consoante os requisitos definidos.

4.6.1. Protótipo 1

O primeiro protótipo, demonstrado na Figura 4.14, e na Tabela 4.2, foi feito com um Arduino UNO R3, e serviu simplesmente para fazer um teste inicial do uso do sensor de temperatura e humidade, juntamente com um ecrã para visualizar os valores lidos.

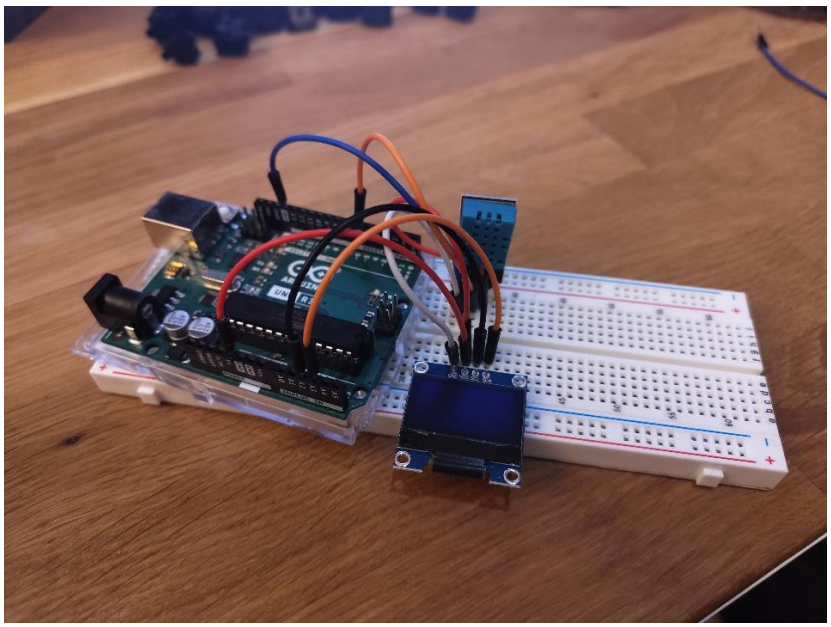


Figura 4.14 - Primeiro protótipo

Tabela 4.2 - Hardware do primeiro protótipo

Microcontrolador	Arduino UNO R3
Sensor de Temperatura e Humidade	DHT11
Ecrã	0.96 Inch OLED 128x64px

A maior limitação deste protótipo inicial era a falta de Wi-Fi, o que não permitia que os dados recolhidos pelos sensores fossem enviados para a base de dados como planeado.

4.6.2. Protótipo 2

No segundo protótipo, demonstrado na Figura 4.15, e na Tabela 4.3, foram feitos alguns *upgrades*, nomeadamente o microcontrolador, que passou de um Arduino UNO R3 para um Arduino Nano33 IoT. Este novo microcontrolador com Wi-Fi integrado, abriu a possibilidade de fazer a ligação a uma rede com acesso à internet e enviar os dados recolhidos pelos sensores para uma base de dados. Foi também adicionado um novo sensor, para medir o nível de concentração de dióxido de carbono, e uma pequena coluna para produzir notificações sonoras.

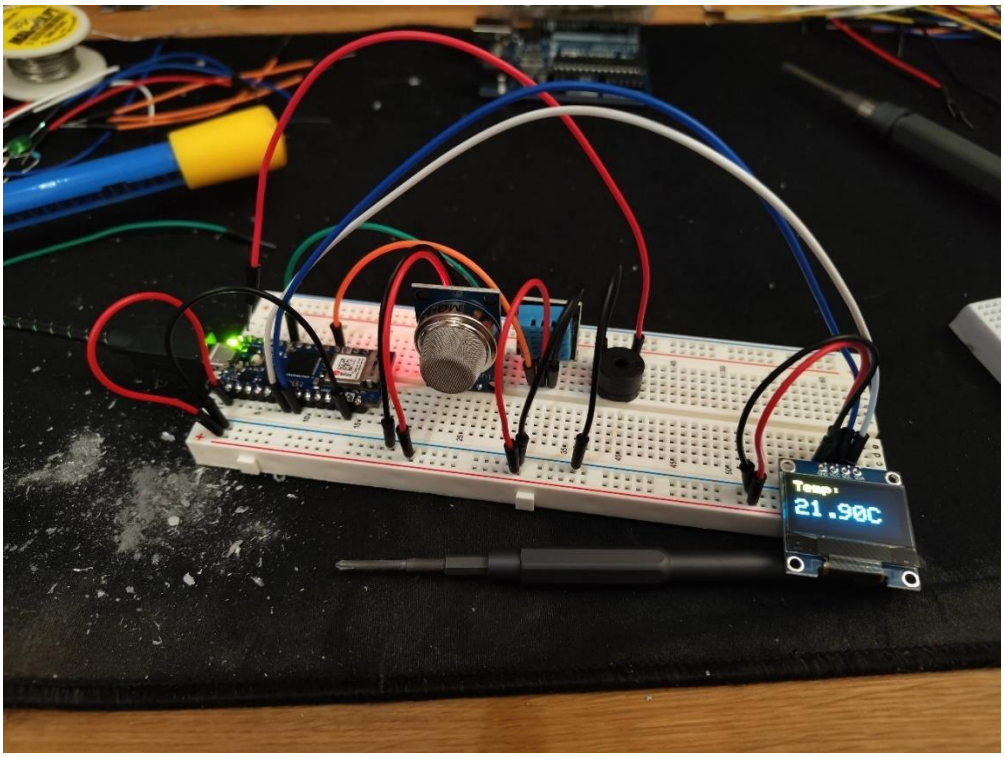


Figura 4.15 - Segundo protótipo

Tabela 4.3 - Hardware do segundo protótipo

Microcontrolador	Arduino Nano 33 IoT
Sensor de Temperatura e Humidade	DHT11
Ecrã	0.96 Inch OLED 128x64px
Sensor de dióxido de carbono	MQ-135
Coluna	Passive buzzer 16 ohms 3V

4.6.3. Protótipo 3

A maior diferença entre o segundo e o terceiro protótipo foi o microcontrolador. No terceiro protótipo trocou-se para o ESP32 devido a ser uma opção com funcionalidades semelhantes ao Arduino Nano 33 IoT, mas muito mais barato. Visto que um dos aspetos importantes do projeto era a relação qualidade/preço dos diversos componentes, a troca para o ESP32 foi uma escolha fácil. O ESP32 acabou por ser o microcontrolador usado na versão final do sistema.

Neste terceiro protótipo foi também trocado o sensor de temperatura e humidade DHT11, para o BME680. Para além de ter medições mais precisas comparativamente ao DHT11, este sensor permite também medir a pressão barométrica e o VOC. Foram também adicionados três LEDs juntamente com três resistências. Estes leds permitem dar um *feedback* visual ao utilizador acerca do nível de dióxido de carbono. A Figura 4.16, e a Tabela 4.4 apresentam o progresso obtido neste terceiro protótipo.

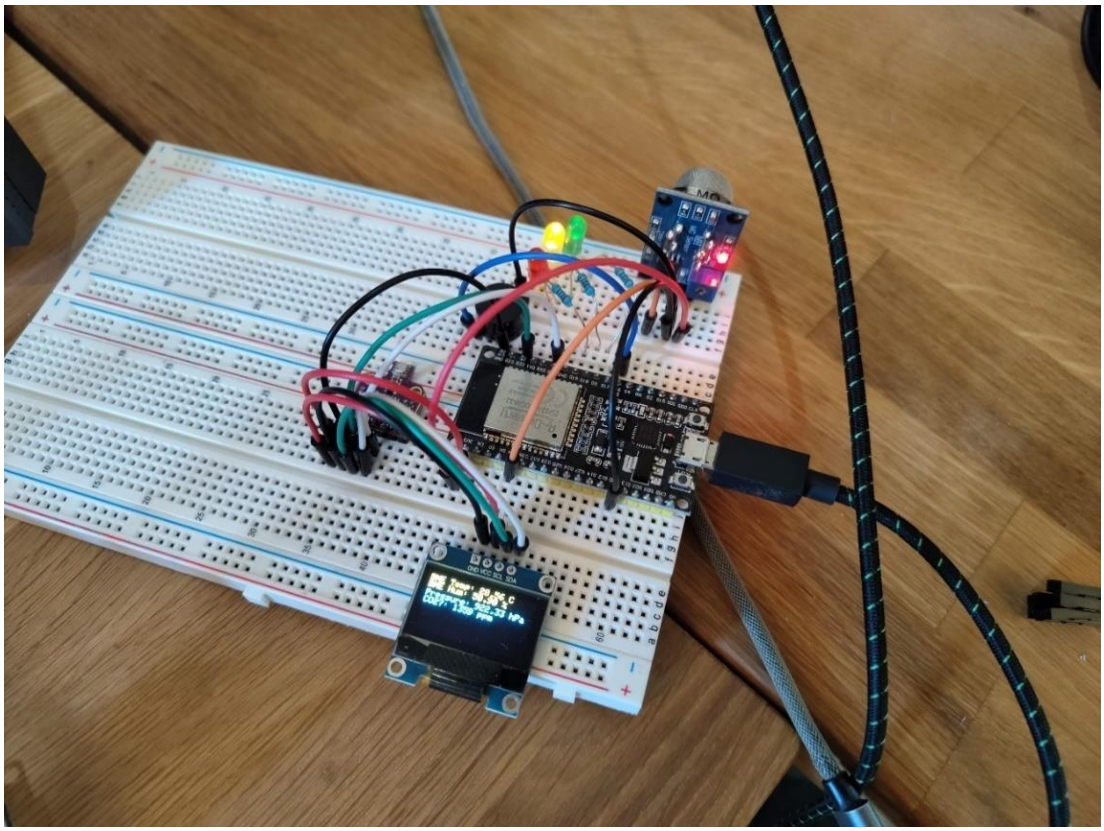


Figura 4.16 - Terceiro protótipo

Tabela 4.4 - Hardware do terceiro protótipo

Microcontrolador	ESP32 Wroom DevKit V1
Sensor de Temperatura, Humidade, Pressão barométrica, VOC	BME680
Ecrã	0.96 Inch OLED 128x64px
Sensor de dióxido de carbono	MQ-135
Coluna	Passive buzzer 16 ohms 3V
LEDs	3d LEDs
Resistências	3x resistências 2k ohms

4.6.4. Protótipo 4

O quarto, e último protótipo, foi o protótipo com a maior quantidade de diferenças em relação ao protótipo anterior. Foram finalmente adicionados os sensores de dióxido de carbono, e de partículas finais.

Para além da adição destes doi sensores, foram trocados os sensores atuais para alternativas superiores. Devido a problemas com medidas do VOC, o BME680 foi trocado por uma alternativa mais barata, o BME280. Visto que o BME280 não consegue medir o VOC, foi adicionado um sensor SGP40. Em relação ao sensor de dióxido de carbono, o MQ-135 demonstrou-se um sensor fraco em termos de precisão de medidas. Depois de alguns testes com outras alternativas como o mhz-19, e o Senseair s8, foi finalmente escolhido o SCD41, que é uma opção fácil de implementar e com um nível de precisão alto. Foi adicionado um *rotary encoder* para que fosse possível alterar a taxa de atualização dos sensores. O ecrã foi trocado para outro modelo maior e com mais resolução, o que permitiu que fosse possível mostrar mais informação ao utilizador. Na Figura 4.17 e na Tabela 4.5 apresenta-se o quarto protótipo do sistema.

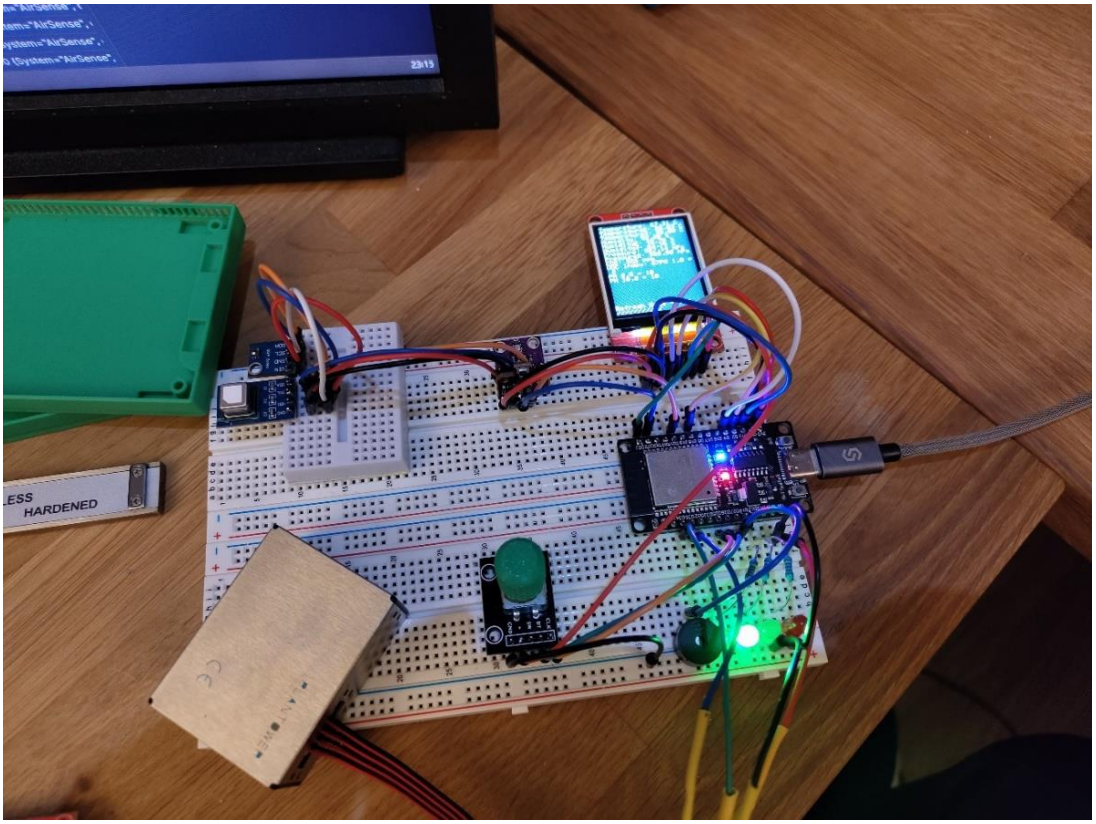


Figura 4.17 - Quarto protótipo

Tabela 4.5 - Hardware do quarto protótipo

Microcontrolador	ESP32 Wroom DevKit V1
Sensor de Temperatura, Humidade, Pressão barométrica	BME280
Ecrã	1.8 Inch TFT LCD 160x128px
Sensor de dióxido de carbono	SCD41
Coluna	Passive buzzer 16 ohms 3V
LEDs	3d LEDs
Resistências	3x resistências 2k ohms
Sensor de partículas	PMS5003
Sensor de VOC	SGP40
Rotary encoder	Rotary Encoder

Por fim, foi definido o *layout* final dos componentes como demonstrado na Figura 4.18.

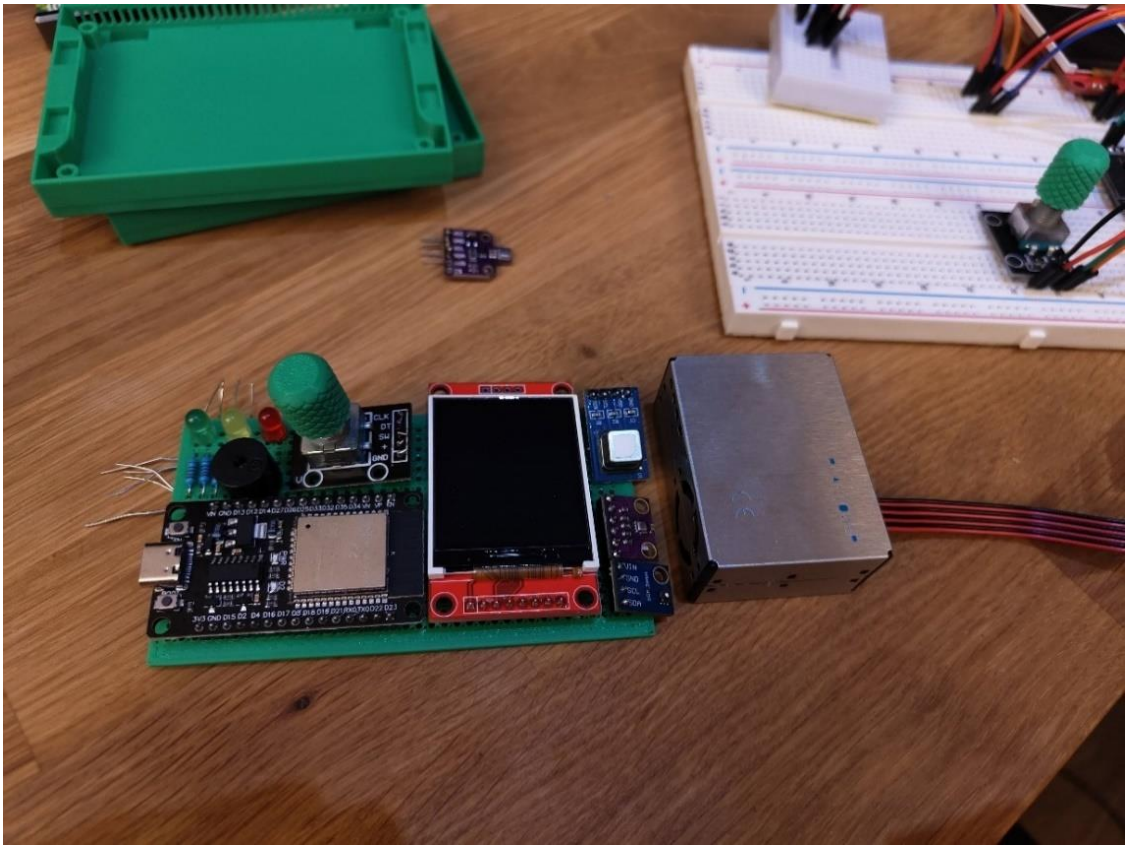


Figura 4.18 - Layout dos componentes do protótipo final

4.7. Protótipo Final

Com tudo montado, obteve-se o resultado demonstrado na Figura 4.19. Os componentes finais estão listados na Tabela 4.6, juntamente com o preço aproximado de cada componente. Como demonstrado na Tabela 4.6, o custo total aproximado do sistema é de 63€, onde a maioria do custo é usado para os sensores de CO2 e de partículas.

Figura 4.19 - Hardware final

Tabela 4.6 - Componentes finais

Componente	Descrição	~ Custo
ESP32 – DevKit 1	Microcontrolador	4€
SCD41	Sensor CO2	21€
SGP40	Sensor VOC	6.3€
BME280	Sensor Temp, Hum. Press.	2.5€
PMS5003	Sensor PM	16€
Coluna	Active buzzer 3V	1€
3x LEDs	Verde, Amarelo, Vermelho	0.1€
Ecrã TFT	1.8 inch TFT	2€
3x Resistências	2kohm	0.03€
Rotary Encoder	Modelo: ky-040	0.5€
PCB	Fabricada na JLCPCB, quantidade mínima: 5	5€
Caixa	Impressa em PLA	1€
Suporte vertical	Impressa em PLA	4€
	Total:	**63.43€**

5. Software do Sistema

Neste capítulo será explicado o *software* do projeto, e como o mesmo interage com os diferentes elementos do *hardware*.

A Figura 5.1 representa a arquitetura do *software*. O microcontrolador envia os dados recolhidos pelos sensores por Wi-Fi para uma base de dados InfluxDB. Os dados guardados na base de dados são depois acedidos pelo Grafana, que serve como interface gráfica para o utilizador, onde se geram todos os gráficos e tabelas com os dados dos sensores. O InfluxDB e o Grafana funcionam ambos na *cloud*, o que permite que o utilizador tenha acesso aos dados em qualquer rede ou computador, em qualquer lado.

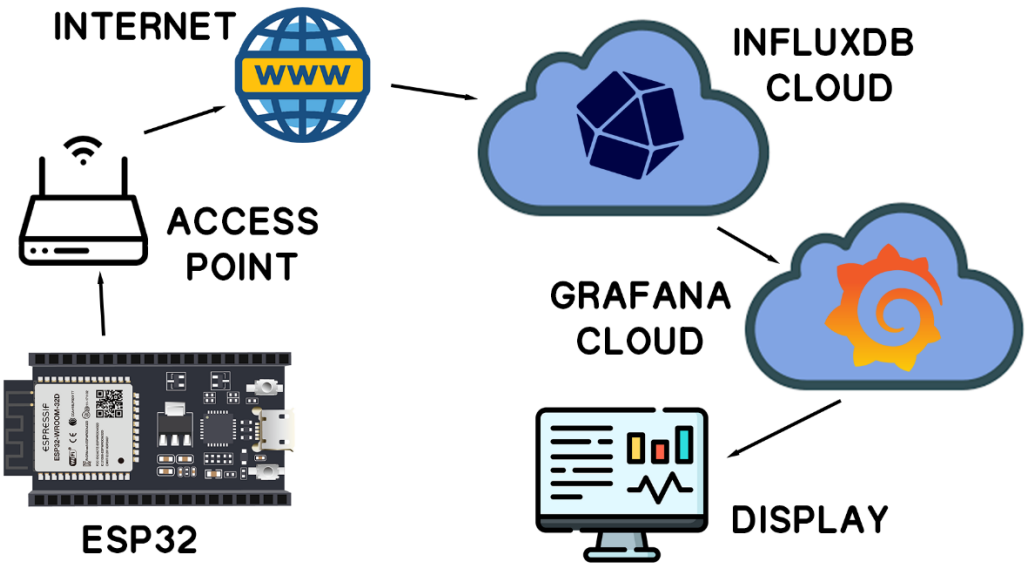


Figura 5.1 - Arquitetura do software

5.1. Funções

De seguida apresentam-se as várias funções desenvolvidas para o sistema:

Função setup():

Esta função corre uma vez quando o sistema liga e serve principalmente para fazer a ligação por Wi-Fi ao ponto de acesso e à base de dados, e para inicializar todos os sensores.

A função começa por gerar um *beep* com a coluna, inicializa o *serial monitor* usado para *debugging* e desenha o ecrã inicial da Figura 5.2.

```
void setup() {
  beep(1);//one beep on startup
  Serial.begin(115200);


  tft.init(); //initialize tft display
  tft.fillScreen(TFT_BLACK);
  tft.setCursor(0, 50, 4);
  tft.setTextColor(TFT_WHITE);
  tft.fillRectVGradient(0, 0, 128, 160, TFT_ORANGE, TFT_RED);
  tft.setRotation(0); //set display rotation
  tft.println("Air\nSense");
  tft.setTextFont(2),
  tft.println("V1.0");
```

Figura 5.2 - Ecrã inicial do sistema

De seguida, é feita a ligação Wi-Fi com o WifiManager. Depois de verificar se a ligação teve sucesso, faz-se a ligação à base de dados.

```
    // reset settings - wipe stored wifi credentials for testing
    //wm.resetSettings();

    WiFiManagerParameter system_name("system_name", "System Name",
"AirSense", 20);
    wm.addParameter(&system_name);
    wm.setTimeout(120);
    bool res;
    res = wm.autoConnect("AirSenseAP");

    if(!res) Serial.println("cant connect to wifi");
    else {
      Serial.print(system_name.getValue());
      Serial.println(" connected to wifi :)");
      beep(2);//two beeps when connected to wifi
    }

    // Connect to InfluxDB
    if (client.validateConnection()) {
      beep(3);//three beeps when connected to db
      Serial.print("Connected to InfluxDB: ");
Serial.println(client.getServerUrl());
    } else Serial.print("InfluxDB ");
Serial.println(client.getLastErrorMessage());

    // Add tags to the data sent do db
    sensor.addTag("device", DEVICE);
    sensor.addTag("System", system_name.getValue());
```

Inicialização dos sensores, do *encoder*, do monitor *serial* usado para o sensor de partículas, e dos pinos usados para os *leds* e o *buzzer*.

```
  Wire.begin();
  //bme680.begin(0x77);
  bme280.begin(0x76);
  scd41.begin(true, true);
  sgp40.begin();


  ESP32Encoder::useInternalWeakPullResistors=UP;
  // use pin 19 and 18 for the first encoder
  encoder.attachHalfQuad(26, 25);
```

```
  // set starting count value after attaching
  encoder.setCount(30);


  Serial1.begin(9600, SERIAL_8N1, RXD2, TXD2);//serial just for pms5003
  if (!Serial1) beep(3);



  pinMode(RED_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);//buzzer

  tft.fillScreen(TFT_BLACK);
}
```

Função loop():

A função *loop* é uma das duas funções principais do sistema. Esta função é executada continuamente milhares de vezes por segundo enquanto o sistema está ligado.

No início da função, verifica-se o valor atual no *rotary encoder*. Como a função UpdateFrequency funciona com valores entre 0 e 60, o valor do *encoder* não pode estar fora deste intervalo.

```
void loop() {

  if(encoder.getCount()>60){
    encoder.setCount(60);
  }else if(encoder.getCount()<0){
    encoder.setCount(0);
  }
```

De seguida, são lidos os valores do sensor de partículas. Como o sensor de partículas funciona de maneira diferente dos outros sensores, os valores têm de ser lidos continuamente e sem interrupções.

Os valores são lidos e colocados em variáveis, que mais tarde são utilizadas nas funções *printtodisplay*, *printtoserial*, e *pushtodatabase*.

```
  if (readPMSdata(&Serial1)){ //pms5003 is a little bitch and needs to
read constantly or else it will throw a checksum error
    particles03um = data.particles_03um;
    particles05um = data.particles_05um;
```

```
    particles10um = data.particles_10um;
    particles25um = data.particles_25um;
    particles50um = data.particles_50um;
    particles100um = data.particles_100um;
    pm10standard = data.pm10_standard;
    pm25standard = data.pm25_standard;
    pm100standard = data.pm100_standard;
    }
```

Com este pedaço de código, é possível correr o que está dentro da instrução *if* a cada 33 milissegundos, ou a 30hz, sem a necessidade de usar *interrupts*. Dentro desta instrução *if*, é impressa a *string* com a taxa de atualização do sistema.

```
    encodercurrentMillis = millis();
    if(encodercurrentMillis - encoderpreviousMillis > 33L){
      encoderpreviousMillis = encodercurrentMillis;

        tft.setCursor(0, 145, 1);
        tft.setTextColor(TFT_WHITE,TFT_BLACK);
        tft.print(UpdateFrequency(encoder.getCount()));
    }
          while(millis()< 15000) interval = 1000UL;
```

Semelhante ao exemplo anterior, usa-se o mesmo princípio para correr algumas funções de x em x tempo consoante o intervalo definido através do *rotary encoder*, o que faz com que seja possível evitar o uso de *interrupts*. Dentro desta instrução *if* é lida a humidade e a temperatura do sensor bme280, e os valores são passados ao sensor SGP40 para que a medida do VOC seja mais precisa. De seguida, são usadas as funções *printtodisplay*, *printtoserial*, *leds*, e *pushtodatabase*.

```
    currentMillis = millis();
    if(currentMillis - previousMillis > interval){
      previousMillis = currentMillis;

      sgp40.setRhT(bme280.readHumidity(), bme280.readTemperature());

      printtodisplay();//prints sensor data to display
      printtoserial();//prints sensor data to serial
      leds();//traffic light leds
      pushtodatabase();//push sensor data to db

  }
}
```

Função printtodisplay():

A função *printtodisplay* gera a interface gráfica para o ecrã TFT demonstrado na Figura 5.3. O método *setCursor* define a posição onde vai ser colocado o texto. Usam-se também os métodos *setTextColor*, *FillRectHGradient*, e *setTextSize* para alterar a cor e o tamanho do texto no ecrã, com o objetivo de criar uma interface mais atraente e fácil de ler.

```
void printtodisplay(){
  tft.setCursor(0, 0, 1);
  tft.setTextColor(TFT_WHITE,TFT_BLACK);
  tft.fillRectHGradient(0, 0, 160, 140, TFT_DARKGREY, TFT_DARKGREY);
  tft.setCursor(1,1);
  //tft.print("Temperature2: "); tft.print(bme280.readTemperature());
tft.println(" C");
  tft.print("Temp:");   tft.setTextSize(2);
tft.print(scd41.getTemperature()); tft.println("
C");  tft.setTextSize(1);
  tft.setCursor(1,21);
  //tft.print("Humidity2: "); tft.print(bme280.readHumidity());
tft.println(" %");
  tft.print("Hum: "); tft.setTextSize(2);tft.print(scd41.getHumidity());
tft.println(" %");tft.setTextSize(1);
  tft.setCursor(1,41);
  tft.print("CO2: "); tft.setTextSize(2);tft.print(scd41.getCO2());
tft.println(" ppm");tft.setTextSize(1);
  tft.setCursor(1,61);
  tft.print("VOC: ");
tft.setTextSize(2);tft.print(sgp40.getVoclndex());tft.println("/500");tft
.setTextSize(1);
  tft.setCursor(1,81);
  tft.print("IAQ: "); tft.setTextSize(2);tft.print("---
");tft.println("/500");tft.setTextSize(1);

  tft.setCursor(1,100);
  tft.println("PM 1.0  = " + String(pm10standard)                    );
  tft.setCursor(1,110);
  tft.println("PM 2.5  = " + String(pm25standard)                    );
  tft.setCursor(1,120);
  tft.println("PM 10.0 = " + String(pm100standard)                   );

  tft.setCursor(1,131);
  tft.print("Pressure: "); tft.print(bme280.readPressure() / 100.0);
tft.println(" hPa");

}
```

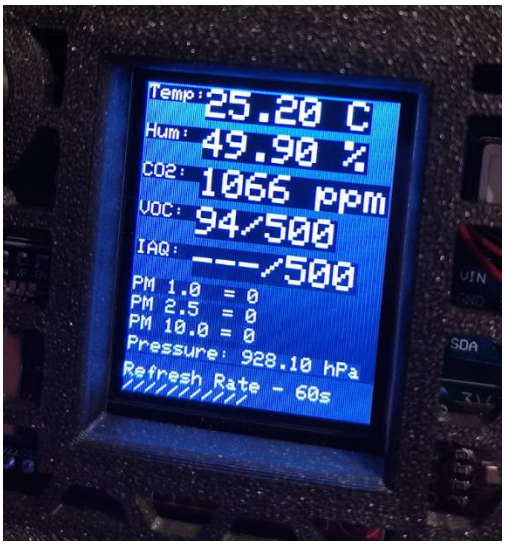


Figura 5.3 - Interface gráfica do ecrã

Função printtoserial():

Esta função é utilizada principalmente para *debugging*, serve simplesmente para imprimir todos os valores lidos dos sensores para o monitor *serial* do IDE.

```
void printtoserial(){
  Serial.println();
  Serial.println(" Refresh Rate = " +
String(interval/1000)                + "s\n");
  //Serial.println("  Temperature = " +
String(bme680.temperature)           + "°C");
  Serial.println(" Temperature2 = " +
String(bme280.readTemperature())     + "°C");
  Serial.println(" Temperature3 = " +
String(scd41.getTemperature())       + "°C");
  //Serial.println("     Humidity = " +
String(bme680.humidity)              + "%");
  Serial.println("    Humidity2 = " +
String(bme280.readHumidity())        + "%");
  Serial.println("    Humidity3 = " +
String(scd41.getHumidity())          + "%");
  //Serial.println("     Pressure = " + String(bme680.pressure /
100)       + " hPa");
  Serial.println("    Pressure2 = " + String(bme280.readPressure() / 100)
+ " hPa");
  Serial.println("          CO2 = " +
String(scd41.getCO2())               + " ppm");
  Serial.println("    VOC Index = " +
String(sgp40.getVoclndex())                );
```

```
  Serial.println("       PM 1.0 = " +
String(pm10standard)                      );
  Serial.println("       PM 2.5 = " +
String(pm25standard)                      );
  Serial.println("      PM 10.0 = " +
String(pm100standard)                + "\n");
  Serial.println("-------------------------------------");
  Serial.print("Particles > 0.3um / 0.1L air:");
Serial.println(particles03um);
  Serial.print("Particles > 0.5um / 0.1L air:");
Serial.println(particles05um);
  Serial.print("Particles > 1.0um / 0.1L air:");
Serial.println(particles10um);
  Serial.print("Particles > 2.5um / 0.1L air:");
Serial.println(particles25um);
  Serial.print("Particles > 5.0um / 0.1L air:");
Serial.println(particles50um);
  Serial.print("Particles > 10.0 um / 0.1L air:");
Serial.println(particles100um);
}
```

Função pushtodatabase():

Esta função começa por limpar os dados dos sensores usados no *upload* anterior.

```
void pushtodatabase(){
  //Clear fields for reusing the point
  sensor.clearFields();
```

De seguida, lê os valores de todos os sensores e adiciona ao ponto "sensor".

```
  // Store measured value into point
  sensor.addField("temperature2", bme280.readTemperature());
  sensor.addField("humidity2", bme280.readHumidity());
  sensor.addField("pressure2", bme280.readPressure() / 100);
  sensor.addField("temperature3", scd41.getTemperature());
  sensor.addField("humidity3", scd41.getHumidity());
  sensor.addField("co2", scd41.getCO2());
  sensor.addField("vocindex", sgp40.getVoclndex());
  sensor.addField("pm10", pm10standard);
  sensor.addField("pm25", pm25standard);
  sensor.addField("pm100", pm100standard);
  sensor.addField("particles03", particles03um);
  sensor.addField("particles05", particles05um);
  sensor.addField("particles10", particles10um);
  sensor.addField("particles25", particles25um);
```

```
  sensor.addField("particles50", particles50um);
  sensor.addField("particles100", particles100um);
  //sensor.addField("airqualityscore", (100-air_quality_score) * 5);
```

No final, verifica se o Wi-Fi continua ligado a alguma rede e se há acesso à base de dados e envia o *point*. Caso o wi-fi não esteja ligado e já tenham passado três horas desde que o sistema está ligado, o sistema é reiniciado para tentar fazer a ligação à rede.

```
  if (client.writePoint(sensor) && WiFi.status() == WL_CONNECTED){
    Serial.print("Writing: "); Serial.println(sensor.toLineProtocol());
  }else if(WiFi.status() == !WL_CONNECTED && millis() > 10800000 ){
    ESP.restart();
  }
}
```

Função beep():

Esta função é utilizada em 3 ocasiões. Quando o hardware é iniciado, quando é feita a ligação a alguma rede por Wi-Fi, e quando há o primeiro acesso à base de dados. Serve para dar um feedback auditivo ao utilizador através da coluna do sistema, e tem também alguma utilidade para *debugging*. A função gera 3 tons auditivos distintos.

```
void beep(int x){
  if(x==1){
    tone(BUZZER_PIN, 500, 200);
  }else if(x==2){
    tone(BUZZER_PIN, 500, 100);
    tone(BUZZER_PIN, 1000, 100);
  }else if(x==3){
    tone(BUZZER_PIN, 500, 75);
    tone(BUZZER_PIN, 1000, 75);
    tone(BUZZER_PIN, 1500, 75);
  }
```

Função leds():

Esta função é responsável pelos 3 *LEDs* de "semáforo" no sistema. Estes 3 *LEDs* são utilizados para representar visualmente o nível de CO2 e informam o utilizador quando o nível de CO2 medido é bom, razoável, ou mau. A função lê o valor de CO2 diretamente do sensor SCD41 e liga ou desliga os leds necessários.

```
void leds(){
  if(scd41.getCO2() >= 1600){
```

```
    digitalWrite(RED_PIN, HIGH); //red led
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(GREEN_PIN, LOW);
  }else if(scd41.getCO2() >= 800 && scd41.getCO2() < 1599){
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, HIGH); //yellow led
    digitalWrite(GREEN_PIN, LOW);
  }else{
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(GREEN_PIN, HIGH); //green led
  }
}
```

Função UpdateFrequency():

Esta função é utilizada para demonstrar a taxa de atualização da leitura dos sensores e do *upload* para a base de dados, esta taxa de atualização é controlada pelo utilizador através do *rotary encoder* no sistema. A função lê o valor atual do *rotary encoder*, e cria uma *string* consoante esse valor. A *string* é devolvida e usada no ecrã TFT do sistema como demonstrado na Figura 5.4.

```
String UpdateFrequency(int encodervalue) {
  String frequencytext = "Refresh Rate - ";
  if       (encodervalue >= 60                       ){
    interval = 300000UL;
    frequencytext += "300s  \n/                     ";
  }else if (encodervalue >= 54 && encodervalue <= 59 ){
    interval = 200000UL;
    frequencytext += "200s  \n///                   ";
  }else if (encodervalue >= 48 && encodervalue <= 53 ){
    interval = 150000UL;
    frequencytext += "150s  \n/////                 ";
  }else if (encodervalue >= 42 && encodervalue <= 47 ){
    interval = 100000UL;
    frequencytext += "100s  \n///////               ";
  }else if (encodervalue >= 36 && encodervalue <= 41 ){
    interval = 80000UL;
    frequencytext += "80s   \n/////////             ";
  }else if (encodervalue >= 30 && encodervalue <= 35 ){
    interval = 60000UL;
    frequencytext += "60s   \n///////////           ";
  }else if (encodervalue >= 24 && encodervalue <= 29 ){
    interval = 50000UL;
    frequencytext += "50s   \n/////////////         ";
  }else if (encodervalue >= 18 && encodervalue <= 23 ){
```

```
    interval = 40000UL;
    frequencytext += "40s   \n///////////////      ";
  }else if (encodervalue >= 12 && encodervalue <= 17 ){
    interval = 30000UL;
    frequencytext += "30s   \n/////////////////    ";
  }else if (encodervalue >= 6  && encodervalue <= 11 ){
    interval = 20000UL;
    frequencytext += "20s   \n///////////////////  ";
  }else if (                    encodervalue <=  5 ){
    interval = 10000UL;
    frequencytext += "10s   \n//////////////////////";
  }
  return frequencytext;
}
```

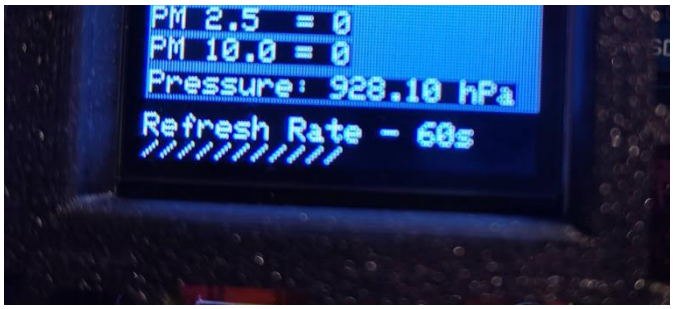


Figura 5.4 - Funcionalidade de alteração da taxa de atualização

5.2. Wi-Fi Manager

Para fazer a ligação do hardware a uma rede de Wi-Fi, implementou-se um sistema de Wi-Fi *Provisioning* com a biblioteca WifiManager. Este sistema elimina a necessidade de ter os dados de acesso a uma rede "*hard-coded*" no código, faz com que se torne mais fácil ligar à rede desejada, e evita a necessidade de se fazer *re-upload* do *firmware* para o microcontrolador cada vez que é preciso trocar de rede.

O WiFiManager é iniciado e depois verifica-se se o sistema ligou ou não a uma rede.

```
    // reset settings - wipe stored wifi credentials for testing
    //wm.resetSettings();

    WiFiManagerParameter system_name("system_name", "System Name",
"AirSense", 20);
    wm.addParameter(&system_name);
    wm.setTimeout(120);
```

```
bool res;
res = wm.autoConnect("AirSenseAP");

if(!res) Serial.println("cant connect to wifi");
else {
  Serial.print(system_name.getValue());
  Serial.println(" connected to wifi :)");
  beep(2);//two beeps when connected to wifi
}
```

Ao iniciar, o sistema cria um ponto de acesso, que pode ser acedido por um smartphone ou computador como demonstrado na Figura 5.5.

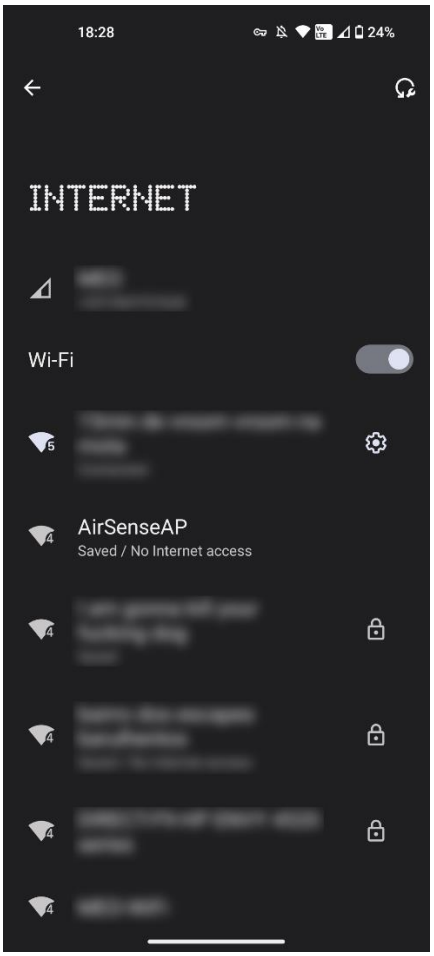


Figura 5.5 - Seleção do access point em Android

Depois da ligação ao ponto de acesso, é aberta automaticamente uma janela no *browser*, onde se pode selecionar o ponto de acesso desejado e onde é inserida a password, como demonstrado na Figura 5.6.

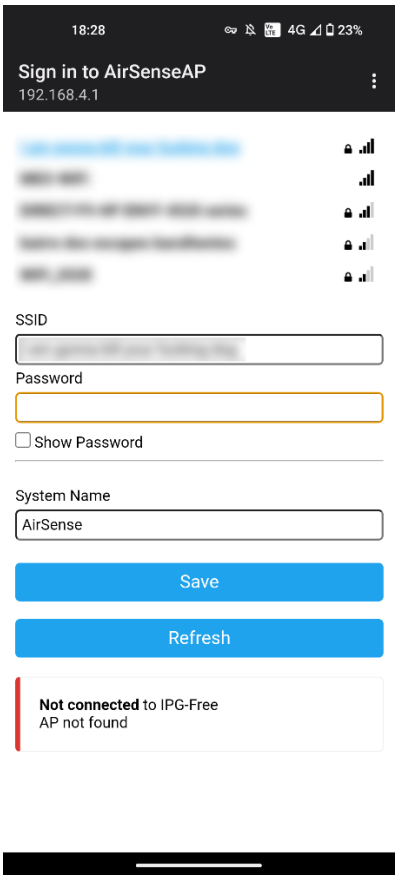


Figura 5.6 - Seleção de rede no WiFiManager

Da próxima vez que o sistema inicia, faz automaticamente a ligação à rede escolhida. Se o sistema iniciar numa zona nova, ou sem acesso a qualquer rede, cria-se de novo o ponto de acesso e é repetido o processo de login. Caso o sistema não tenha acesso a qualquer rede, funciona em modo offline e mostra apenas os dados dos sensores no ecrã integrado.

5.3. InfluxDB

Com acesso à internet, o sistema envia todos os valores recolhidos pelos sensores para uma base de dados de séries temporais InfluxDB. Os valores são armazenados num "*bucket*" que contém todos os valores recolhidos, juntamente com o respetivo *timestamp*. Os dados podem ser acedidos com *queries* em SQL, como demonstrado na Figura 5.7, no entanto, a interface com o sistema é feita maioritariamente pelo Grafana, o que torna a interação com o *frontend* do InfluxDB desnecessário para o utilizador.

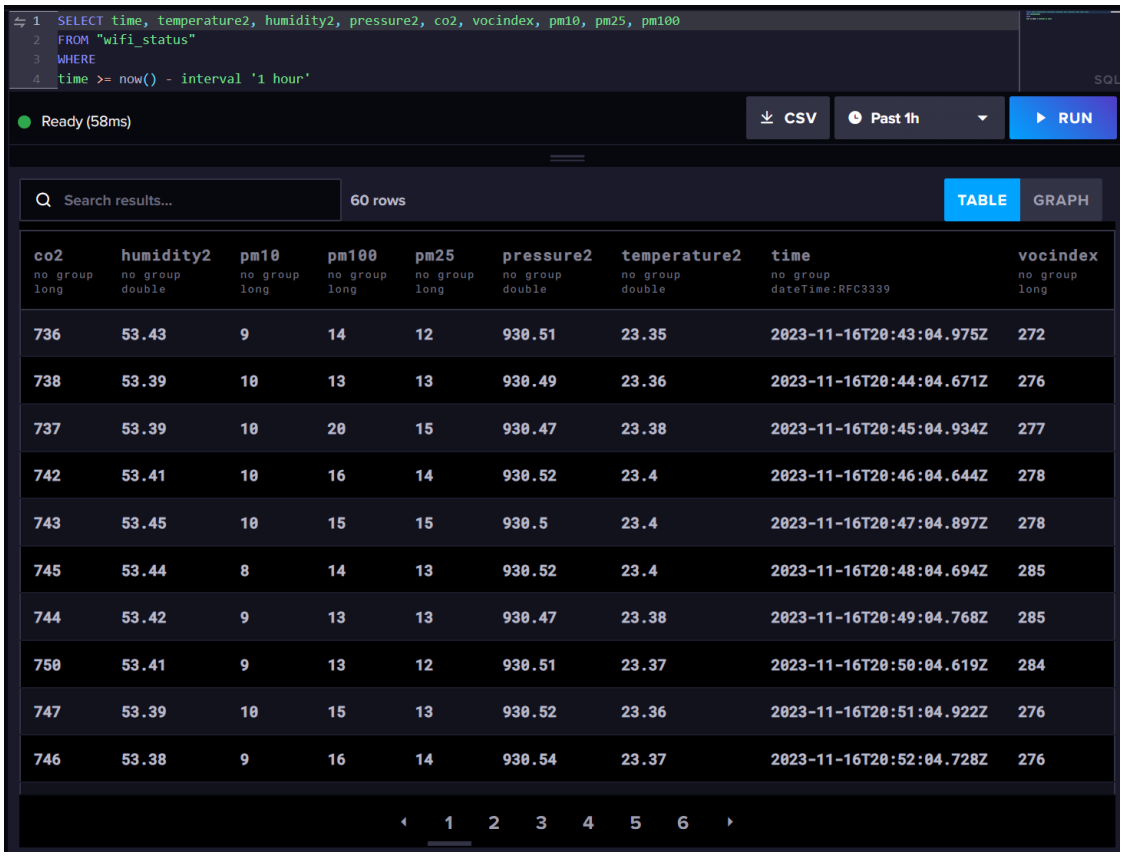


Figura 5.7 - Interface da InfluxDB

A base de dados está armazenada na *cloud* e pode ser acedida em qualquer lado.

5.4. Grafana

Por fim, como *frontend* para o utilizador, foi usado o software Grafana. No Grafana é criada uma *dashboard*, que contém todos os painéis para visualização de gráficos. Cada painel é feito com dois elementos principais:

-Um *query*, usando a linguagem de consulta FLUX da InfluxDB;

-Configuração do painel, com todas as possíveis definições.

Para o *query* de cada painel, é necessário selecionar o *bucket* onde foram armazenados os dados. Neste caso, o *bucket* utilizado chama-se "*humidityandtemperature*", depois escolhe-se um intervalo temporal, começando por "*v.timeRangeStart*" e acabando com "*v.timeRangeStop*". Estas duas variáveis são alteradas no Grafana consoante o intervalo

temporal selecionado pelo utilizador. Por fim, seleciona-se o campo desejado - neste caso o painel é para visualizar a temperatura.

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "temperature"
  )
```

A grande maioria dos painéis usa um *query* como indicado anteriormente. Contudo, foram também implementados outros tipos de painéis, com um *query* diferente, para visualizar os máximos e mínimos diários de cada sensor nos últimos 30 dias. Para esses painéis, são utilizados os seguintes *queries*:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "temperature2")
  |> aggregateWindow(every: 24h, fn: max, createEmpty: false)

from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "temperature2")
  |> aggregateWindow(every: 24h, fn: min, createEmpty: false)
```

Estes *queries* funcionam de maneira semelhante, mas com uma linha adicional que serve para agregar os valores das últimas 24 horas, e determinar o valor máximo e mínimo.

Depois de gerar o *query* desejado, é necessário definir o estilo de gráfico e configuração mais apropriada para cada uma das medidas, escolhendo entre as múltiplas opções disponíveis no Grafana.

A interface final é demonstrada nas figuras seguintes. Na Figura 5.8 é possível ver os gráficos temporais para temperatura, humidade, pressão, CO2, e VOC, assim como os indicadores para o valor atual de CO2 e VOC.

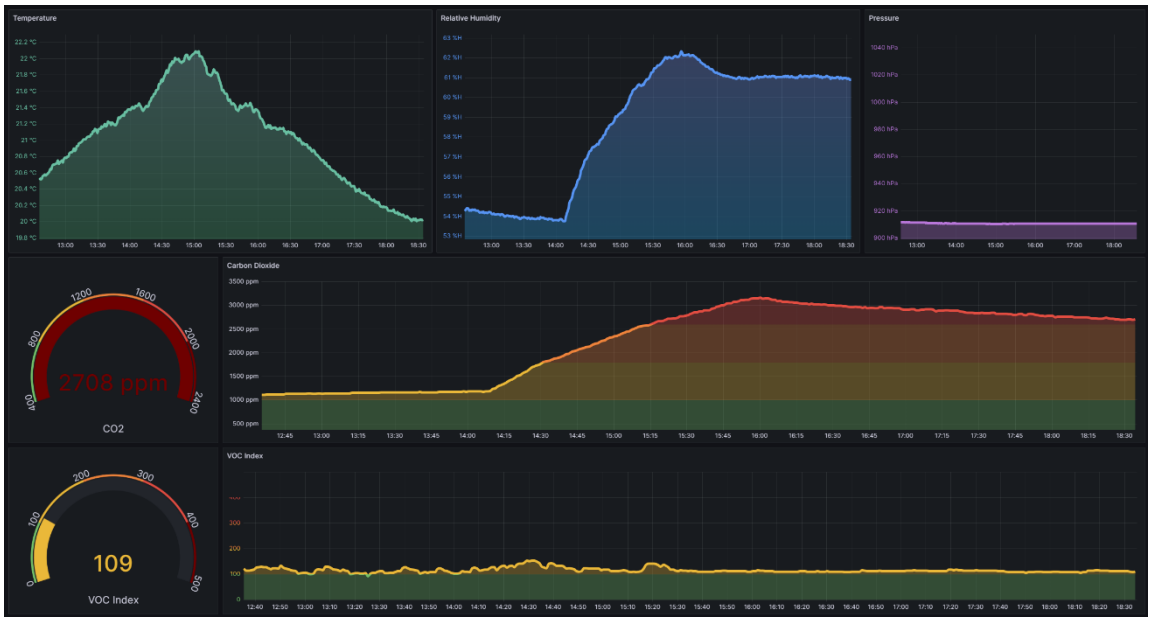


Figura 5.8 - Gráficos de temperatura, humidade, pressão, co2, e VOC Index

A Figura 5.9 contém os gráficos do nível de partículas: o primeiro gráfico representa os valores de PM 1.0, PM2.5, e PM10.0 ao longo do tempo, enquanto que o segundo e o terceiro gráficos representam os níveis de partículas por cada 0.1L de ar atual, e em histórico temporal.

Figura 5.9 - Gráfico temporal do nível de partículas

A Figura 5.10 ilustra as tabelas dos valores máximos e mínimos de temperatura, humidade, CO2, e VOC, nos últimos 30 dias.

Figura 5.10 - Valores máximos e mínimos diários

Por fim, a Figura 5.11, e a Figura 5.12 ilustram todos os gráficos referidos anteriormente mas em layout para um *smartphone*.

Figura 5.11 - Layout mobile

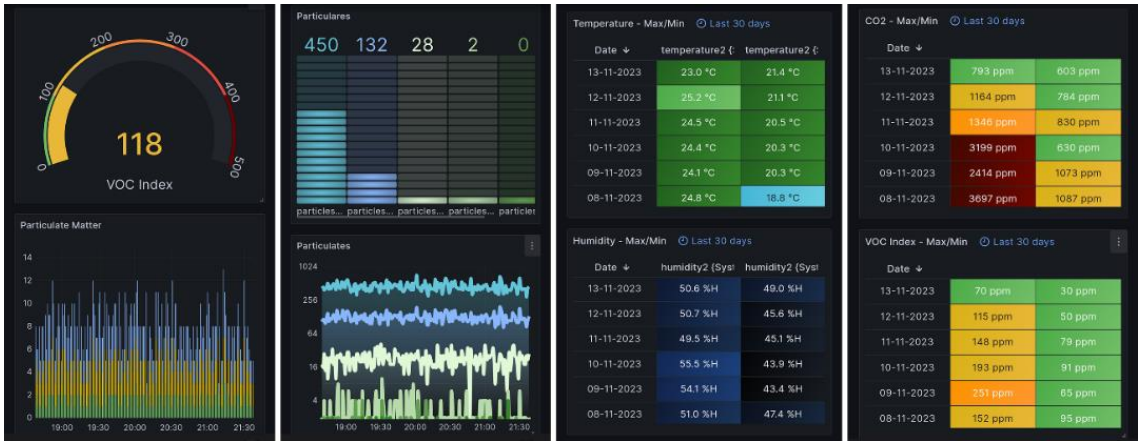


Figura 5.12 - Layout mobile 2

6. Verificação e Testes

Com o objetivo de verificar os valores registados pelo sistema, foi realizado um teste ao longo de 17 dias.

Este teste foi feito na Escola Superior de Tecnologia e Gestão do IPG, na sala 39, um laboratório de informática, entre os dias 30 de outubro e 15 de novembro. Dado o ambiente selecionado para esta validação, o teste focou-se principalmente nos valores de concentração de CO2.

Apesar de alguns problemas com a ligação à rede do IPG, foi possível isolar certos intervalos de tempo com leituras contínuas, e verificar que o sistema consegue medir diferenças em concentração do nível de CO2 corretamente. Estas diferenças em CO2 são causadas por diferentes fatores, como, o número de alunos presentes na sala, ou o nível de ventilação na sala.

No ponto seguinte será feita uma análise a alguns destes intervalos de leituras, justamente com uma explicação acerca do que pode causar o aumento, ou decréscimo dos valores registados nos exemplos em questão. Para solidificar a análise de alguns dos seguintes exemplos, foram anotados certos fatores pelo professor Celestino Gonçalves, como o horário de certas aulas, assim como o número de pessoas presentes, e as ocasiões em que a sala foi arejada.

Foram também feitos alguns testes adicionais fora do IPG para verificar o funcionamento dos outros sensores do sistema.

6.1. Análise de Resultados

Testes do sensor de CO2:

Em relação às leituras de concentração de CO2, foi possivel, em certos casos, detetar o seguinte:

- Quando os alunos entram ou saem da sala;
- Se a sala tem muitos, ou poucos alunos;

- Quando a sala é arejada.

No exemplo da Figura 6.1, é possível verificar que segunda-feira, dia seis de novembro há um aumento rápido entre as 09:30 e as 11:30, seguido de um decréscimo entre a as 11:30 e as 14:30, e por fim, outra subida, um pouco menos drástica (devido a ter apenas 5 pessoas) que a anterior, entre as 14:30 e as 16:30. Ambas as subidas coincidem com aulas na sala 39, como é possível verificar no horário na Figura 6.2. Depois do final da aula, há uma descida gradual até à manhã do dia seguinte.

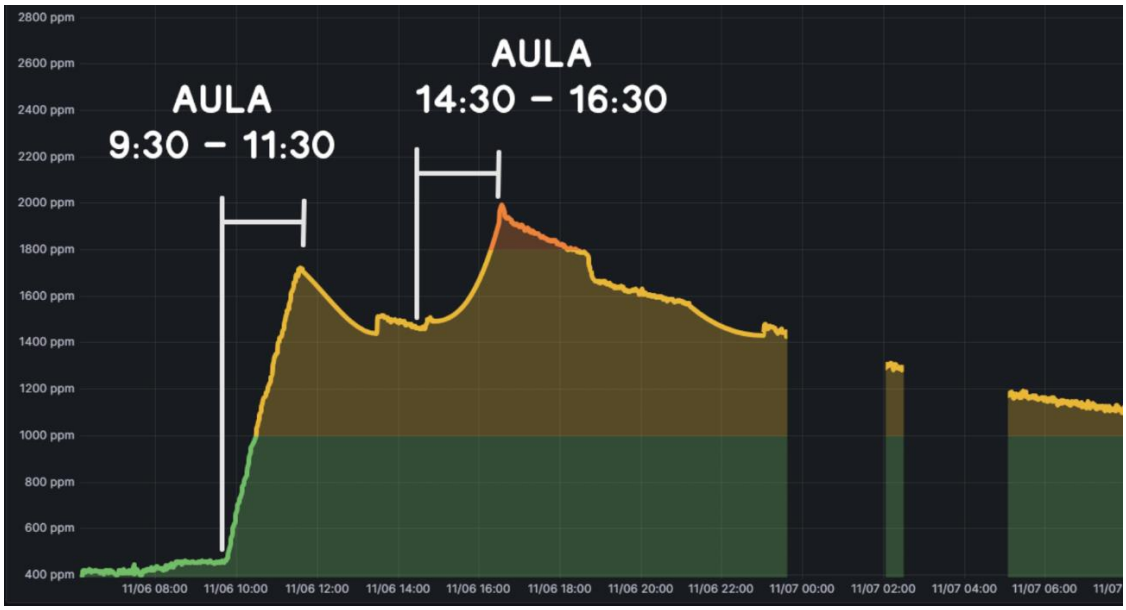


Figura 6.1 - Segunda-feira dia seis de novembro

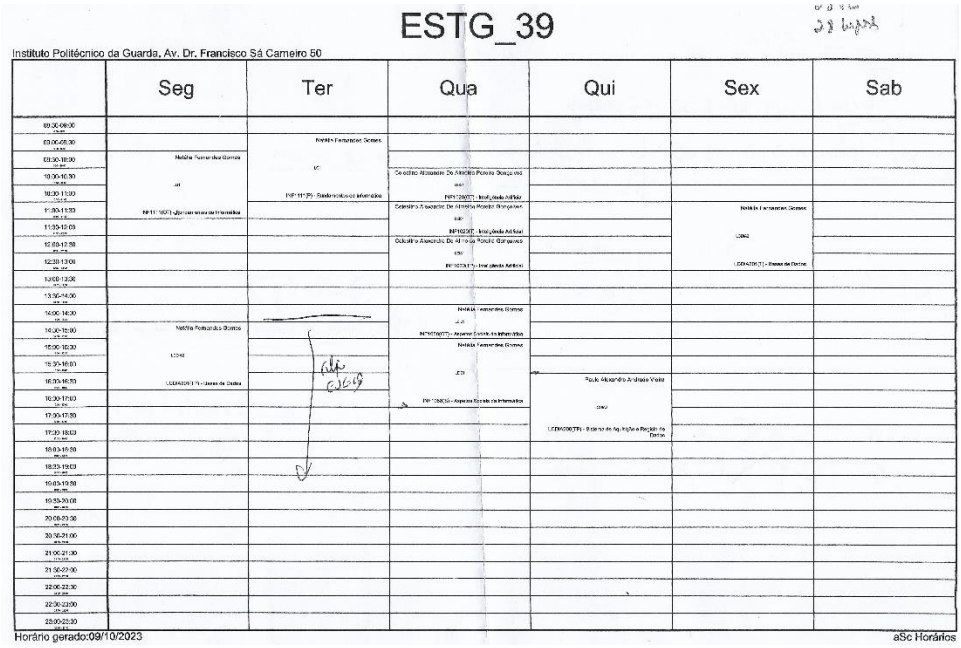


Figura 6.2 - Horário da sala 39, ESTG

Terça-feira, dia 7 de novembro, demonstrado na Figura 6.3, é possível identificar a aula que decorreu entre as 09:00 e as 11:00. Apesar da falha ao início da tarde, vê-se um aumento bastante brusco que começa próximo das 14:00, com um pico às 16:00. Por seguida, é possível identificar uma descida rápida entre as 18:50 e as 19:00, o que representa uma grande probabilidade de que a sala tenha sido ventilada, visto que quando a sala não tem ventilação, o nível de CO2 nunca desce tão bruscamente.

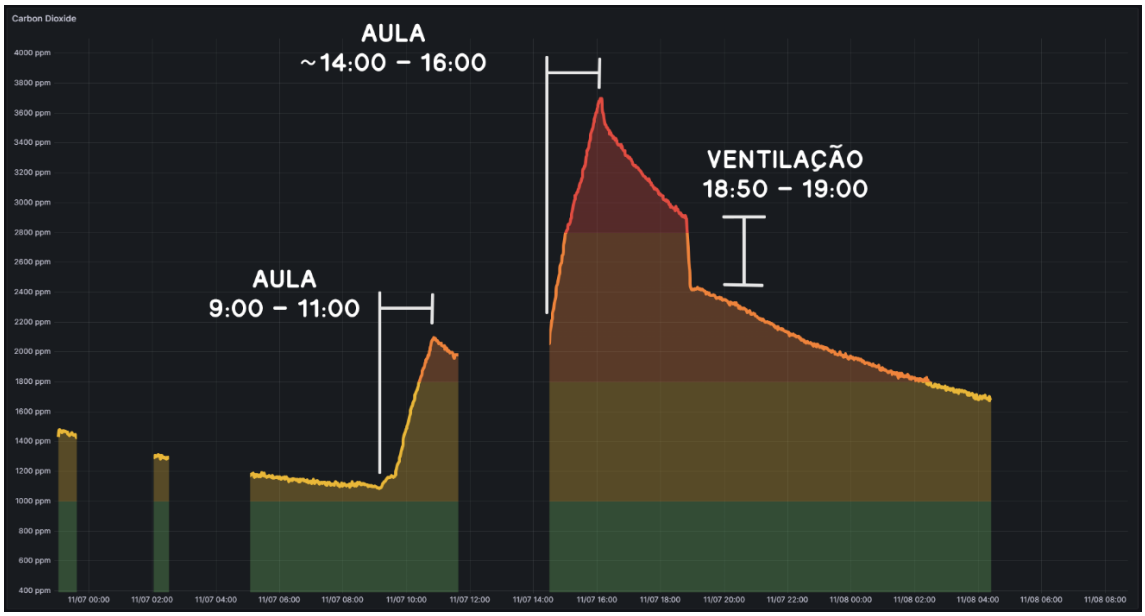


Figura 6.3 - Terça-feira dia sete do novembro

Dia nove de novembro, uma quinta-feira, demonstrado na Figura 6.4, o sistema deteta uma possível aula das 14:00 às 16:00 com bastantes alunos (devido ao declive alto do gráfico), que não consta no horário da sala 39. É possível que a sala tenha sido arejada a partir das 15:50 dada a descida brusca do nível de concentração de CO2 durante as três horas seguintes.

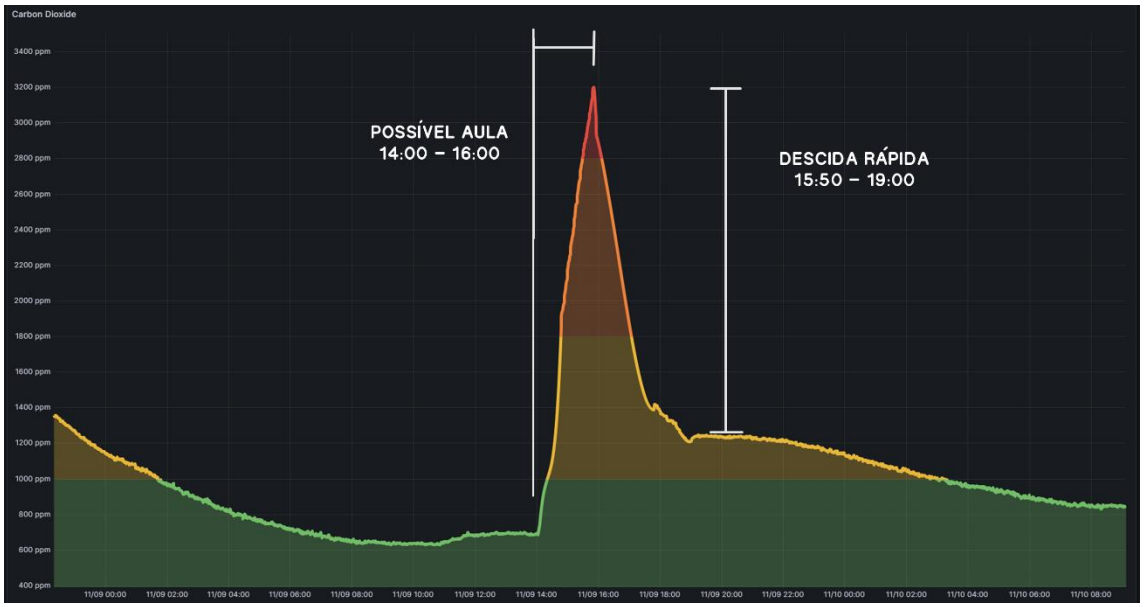


Figura 6.4 - Quinta-feira dia nove de novembro

Na Figura 6.5, demonstra-se o dia 10 de novembro, uma sexta-feira. É possível ver o aumento de CO2 durante a aula das 11:00 às 13:00. De seguida há um período de 15 minutos em que a sala é ventilada pelo professor Celestino Gonçalves, das 16:45 às 17:00. A concentração de CO2 desce gradualmente durante o fim de semana.

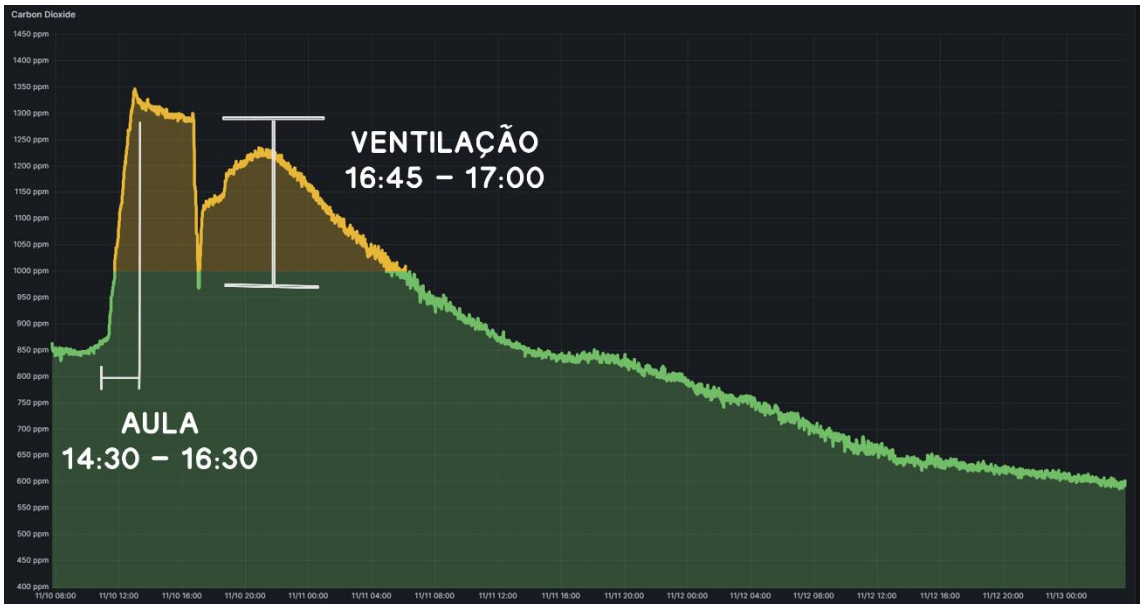


Figura 6.5 - Sexta-feira dia 10 de novembro

Na segunda-feira, dia 13 de novembro, demonstrado na Figura 6.6, é possível identificar as mesmas aulas da semana anterior. Depois da descida gradual durante o fim de semana,

há um aumento rápido entre as 09:30 e as 11:30, seguido por um intervalo de 3 horas e meia, e por fim, um pequeno aumento na aula das 14:30 às 16:30 com 5 pessoas.

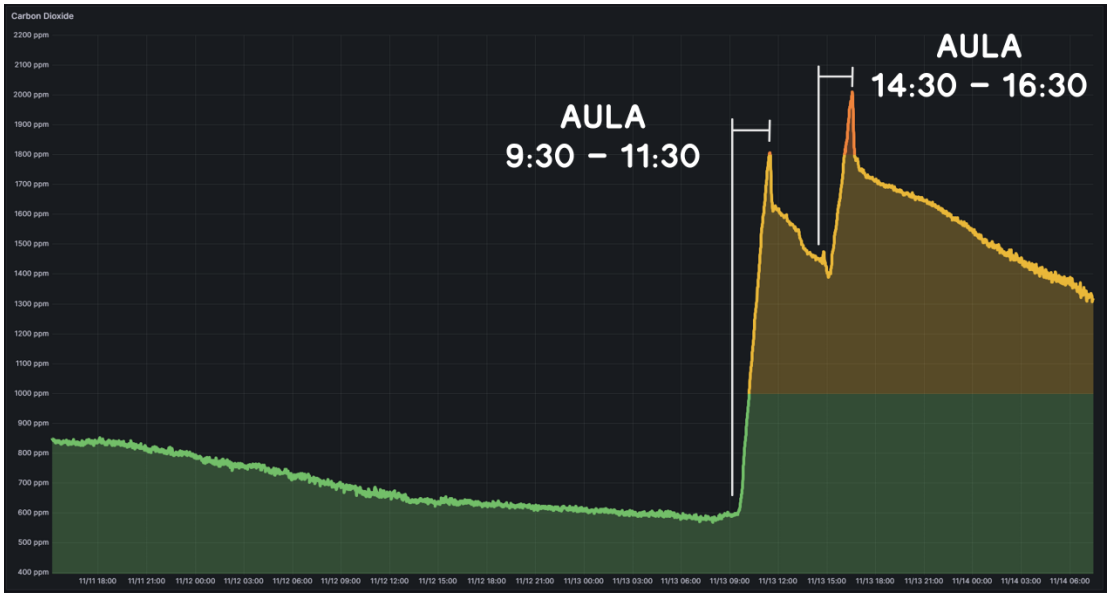


Figura 6.6 - Segunda-feira dia 13 de novembro

Dia 15 de novembro, quarta-feira, demonstrado na Figura 6.7 decorre uma aula com 24 alunos. Apesar do nível elevado de alunos, devido à janela ter ficado aberta, o valor de concentração do CO2 mantem-se num nível bom durante a maioria da duração da aula.

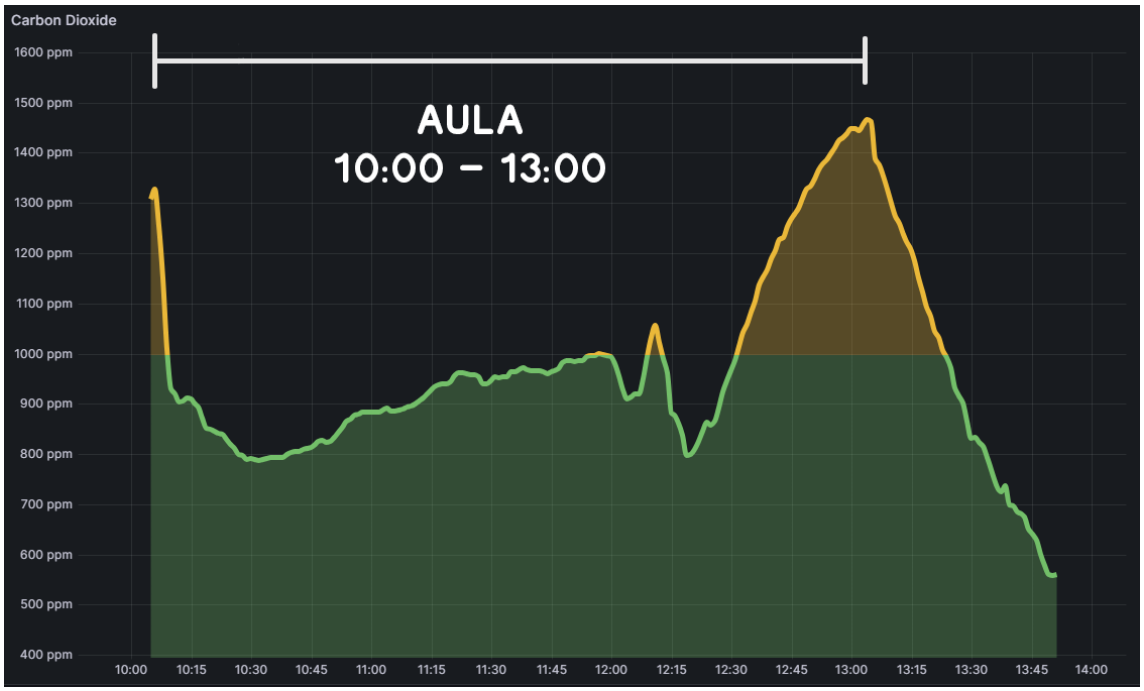


Figura 6.7 - Quarta-feira dia 15 de novembro

Testes do sensor de temperatura:

Em relação à temperatura, foi possível verificar que o sensor deteta a mudança de temperatura do sistema de aquecimento da escola, na Figura 6.8 é possível visualizar a temperatura a aumentar e diminuir em intervalos constantes. Entre os dias 10 e 14 de novembro, repete-se um padrão de aumento e diminuição da temperatura, em que aumenta sempre das 08:00 às 12:00, e outra vez das 16:00 às 21:00. No Domingo, dia 13 de novembro, não há esta mudança de temperatura porque a escola está fechada.

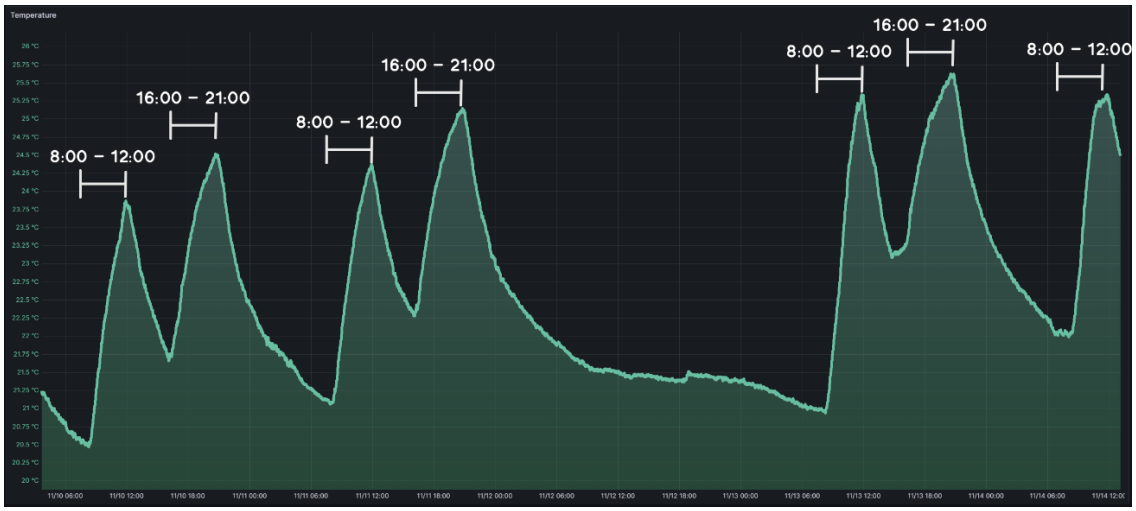


Figura 6.8 - Temperatura, dia 10-14 novembro

Para avaliar o resto dos sensores, foram feitos alguns testes fora da ESTG.

Teste do sensor de humidade relativa:

No teste do sensor de humidade, foi utilizado um desumidificador numa divisão fechada. O desumidificador liga e desliga periodicamente em intervalos de 10 minutos. Na Figura 6.9 é possível visualizar uma descida incremental, juntamente com os intervalos periódicos em que o desumidificador está ligado e desligado.

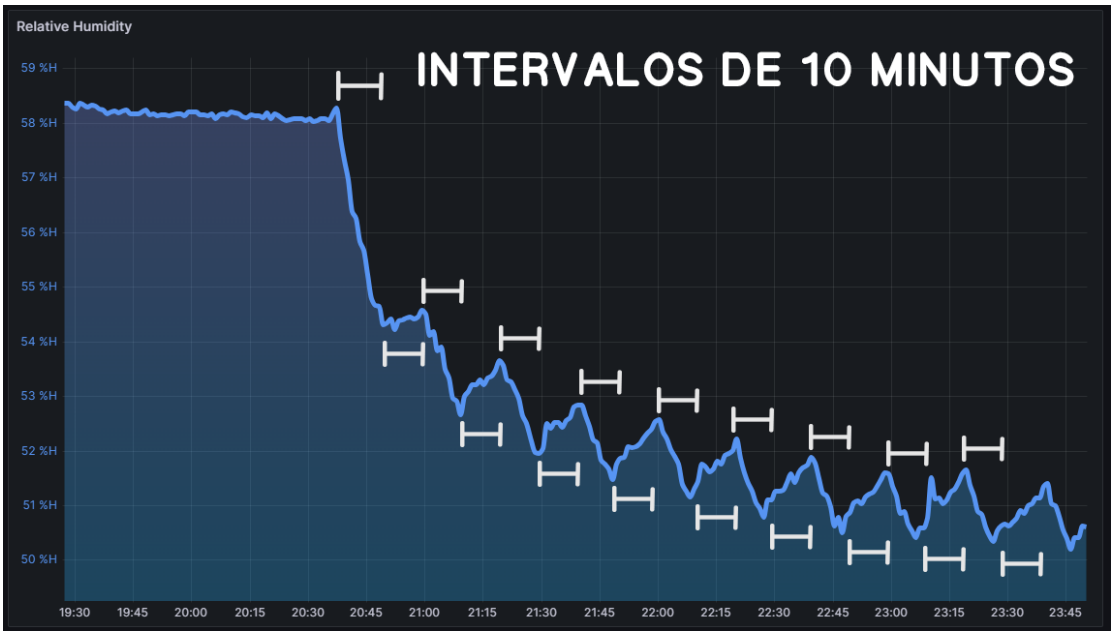


Figura 6.9 - Teste com desumidificador

Teste do sensor de compostos orgânicos voláteis (VOC):

Para testar o sensor de VOC, foi colocado um pano embebido em álcool perto do sistema. O sensor conseguiu detetar o composto volátil e o valor lido disparou de cerca de 100/500 para 450/500 como demonstrado na Figura 6.10.

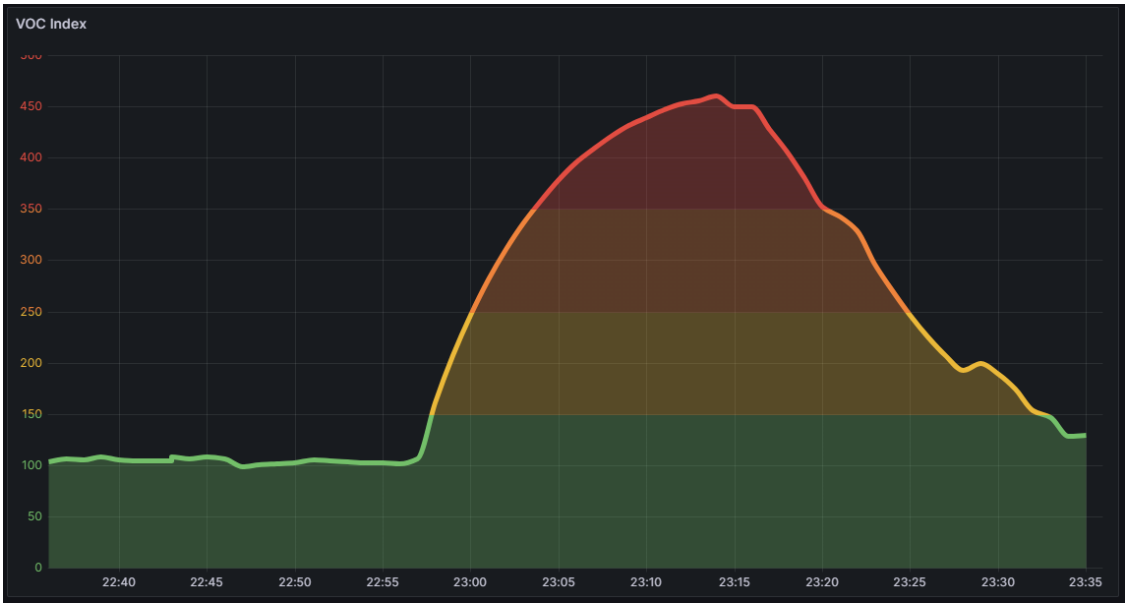


Figura 6.10 - Teste do sensor de VOC

Teste do sensor de partículas:

No teste do sensor de partículas foi colocado um pedaço de cartão queimado numa frigideira perto do sistema. O sensor conseguiu detetar rapidamente as partículas no fumo e os valores passam de cerca de 1-10 (habitual para um ambiente limpo), para a casa dos milhares como se pode ver na Figura 6.11.

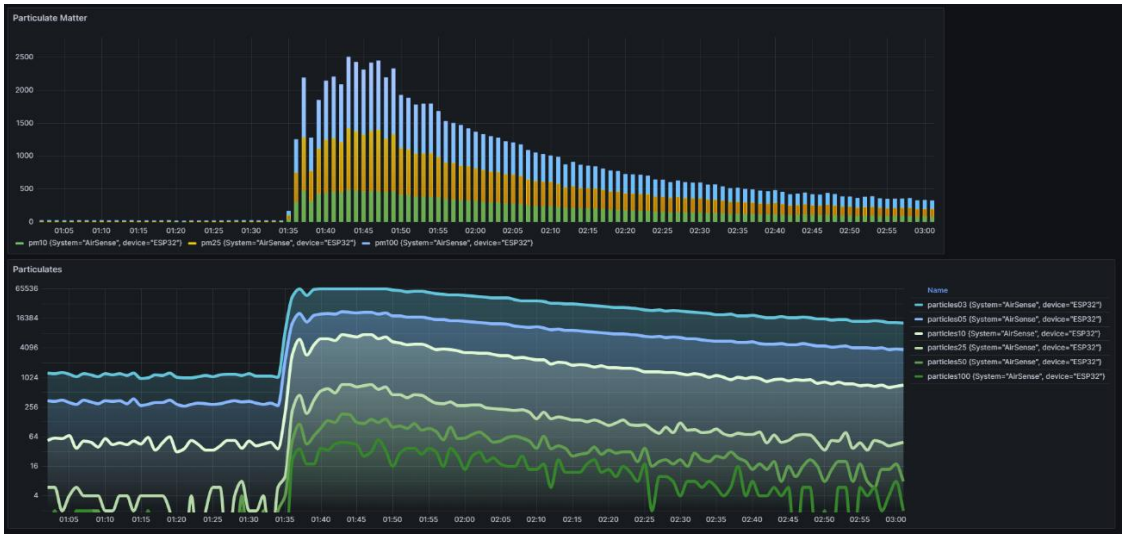


Figura 6.11 - Teste do sensor de partículas

7. Conclusão

Este projeto teve como objetivo o estudo, análise, e construção de um sistema de monitorização da qualidade do ar interior. Durante a análise inicial ao tema, foi feito o estudo e comparação de vários projetos existentes semelhantes e soluções comerciais disponíveis atualmente no mercado. Foi também realizado o estudo de várias áreas relativas ao projeto que eram previamente pouco conhecidas pelo aluno, como modelação 3D, desenvolvimento de placas de circuito impresso (PCB), e implementação de sensores com microcontroladores.

Estes novos conhecimentos foram aplicados ao longo do processo na criação de quatro protótipos com características diferentes, até se ter conseguido desenvolver um protótipo final mais refinado, que conseguiu medir níveis de temperatura, humidade relativa, pressão barométrica, dióxido de carbono, compostos orgânicos voláteis, e partículas. Foram realizados vários testes para validar o funcionamento do sistema final e a exatidão dos valores monitorizados. A maioria dos requisitos propostos inicialmente foram satisfeitos, no entanto, apesar do sistema final ter ficado funcional, e os testes de validação terem tido sucesso, haverá sempre espaço para melhorias e funcionalidades adicionais para implementar no futuro.

O projeto foi concluído com sucesso e os objetivos definidos inicialmente foram alcançados. A experiência adquirida durante o projeto foi gratificante e permitiu consolidar os conhecimentos obtidos ao longo do curso de Engenharia Informática, especialmente das cadeiras de Arquitetura de Computadores, Programação Avançada, e Sistemas Digitais. Foi também possível adquirir novos conhecimentos em relação às áreas de Sistemas Embebidos, Microcontroladores, e Internet das Coisas.

7.1. Trabalho Futuro

De seguida apresentam-se algumas ideias para trabalho futuro e de melhoria do projeto atual:

- Sistema de notificações e alertas por SMS ou e-mail;

- Integração de alguma inteligência artificial, que, com uma análise ao histórico de medidas, consiga fazer certas previsões ou encontrar padrões nos dados fornecidos. Como por exemplo, tentar prever quando é que o CO2 vai chegar a um nível não saudável, ou tentar detetar que num dia específico da semana e a uma certa hora, o nível de VOC aumenta inesperadamente;

- Desenvolver uma segunda versão mais compacta, com uma PCB sem módulos ou *breakout boards*, com os sensores e o microcontrolador soldados diretamente na PCB, juntamente com todos ou outros componentes como resistências ou condensadores;

- Resolver falhas ou limitações do sistema, como por exemplo, não poder selecionar uma rede de Wi-Fi diferente na mesma localização quando o sistema já está ligado a uma rede;

- Implementar um grupo de vários sistemas num espaço como numa casa, um em cada divisão, e criar um mapa onde se possam ver todas as medidas de cada divisão;

- Implementar funcionalidade para o sistema guardar as medidas em memória caso haja falhas no Wi-Fi;

- Criar documentação e disponibilizar todos os ficheiros organizados no GitHub.

Referências

[1] EPA, "Indor Air Quality," [Online]. Available: https://www.epa.gov/report-environment/indoor-air-quality#:~:text=Pollutants%20and%20Sources,-Typical%20pollutants%20of&text=Combustion%20byproducts%20such%20as%20carbon,Pesticides%2C%20lead%2C%20and%20asbestos. [Accessed 11 2023].

[2] EPA, "Indoor Particulate Matter," [Online]. Available: https://www.epa.gov/indoor-air-quality-iaq/indoor-particulate-matter. [Accessed 11 2023].

[3] Government of Canada, "Carbon dioxide in your home," [Online]. Available: https://www.canada.ca/en/health-canada/services/publications/healthy-living/carbon-dioxide-home.html co2. [Accessed 11 2023].

[4] Department of Health, "Carbon Dioxide (CO2)," [Online]. Available: https://www.health.state.mn.us/communities/environment/air/toxins/co2.html. [Accessed 11 2023].

[5] A. Ramos, V. B. Jesus, C. Gonçalves, F. Caetano and C. Silveira, "Monitoring Indoor Air Quality and Occupancy with an IoT System: Evaluation in a Classroom Environment," *2023 18th Iberian Conference on Information Systems and Technologies (CISTI),* p. 6, 2023.

[6] G. Marques and R. Pitarma, "An Internet of Things-Based Environmental Quality," *MDPI,* p. 18, 2019.

[7] H. Fritz, S. Bastami, C. Lin, K. Nweye, T. To, L. Chen, D. Le, A. Ibarra, W. Zhang, J. Y. Part, W. Waites, M. Tang, P. Misztal, A. Novoselac, E. Thomaz, K. Kinney and Z. Nagy, "Design, fabrication, and calibration of the Building EnVironment and," *Building and Environment,* vol. 222, p. 23, 2022.

[8] S. Ward, M. Gittens, N. Rock and K. James, "CampusEMonitor: Intelligent Campus Environment Room," *SIGUCCS '19: Proceedings of the 2019 ACM SIGUCCS Annual Conference,* p. 8, 2019.

[9] P. K. Sharma, A. Mondal,, S. Jaiswal,, M. Saha,, S. Nandi,, T. De and S. Saha, "IndoAirSense: A framework for indoor air quality estimation," *Atmospheric Pollution Research,* vol. 12, no. 1, p. 13, 2021.

[10] B. W. Dionova, D. Hendrawati, M. N. Abdulrazaq, D. J. Vresdian, A. A. Hapsari, M. I. Abdullah and L. P. Pratama, "Design and Simulation of Environment Indoor," *2023 International Seminar on Intelligent Technology and Its Applications (ISITIA),* p. 6, 2023.

[11] J. Rosenberger, Z. Guo,, A. Coffman, D. Agdas and P. Barooah, "An Open-Source Platform for Indoor Environment Monitoring," *Sensors,* p. 17, 2022.

[12] A. Maier, A. Sharp and Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things," *2017 Internet Technologies and Applications (ITA)*, 2017.

[13] Grafana, "About Grafana," [Online]. Available: https://grafana.com/docs/grafana/latest/introduction/. [Accessed 11 2023].

[14] Microsoft, [Online]. Available: https://code.visualstudio.com. [Accessed 11 2023].

[15] Autodesk, "What is Autodesk Fusion 360?," [Online]. Available: https://www.autodesk.com/solutions/what-is-fusion-360#:~:text=Fusion%20360%20is%20an%20integrated,and%203D%20CAD%20drafting%20software.. [Accessed 11 2023].

[16] EasyEDA, "About EasyEDA," [Online]. Available: https://easyeda.com/page/about. [Accessed 11 2023].

[17] Bosch, "BME280 measuring principle," [Online]. Available: https://community.bosch-sensortec.com/t5/MEMS-sensors-forum/BME280-measuring-principle/td-p/7674. [Accessed 11 2023].

[18] Internet of Chemistry Things, [Online]. Available: https://ioct.tech/edu/sites/default/files/2019-04/PMS5003-Educational%20Version%202.pdf. [Accessed 11 2023].

[19] Airgradient, "Low Cost CO2 Sensors Comparison: Photo-Acoustic vs NDIR," [Online]. Available: https://www.airgradient.com/blog/co2-sensors-photo-acoustic-vs-ndir/#:~:text=Photo-acoustic%20sensors%20use%20the,the%20absorption%20with%20a%20microphone.. [Accessed 11 2023].

[20] DFRobot, "Fermion: SGP40 Air Quality Sensor (Breakout)," [Online]. Available: https://www.dfrobot.com/product-2241.html. [Accessed 11 2023].

[21] "ESP32 DevKit ESP32-WROOM GPIO Pinout," [Online]. Available: https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/.

[22] "DIY Air Quality Monitor - PM2.5, CO2, VOC, Ozone, Temp & Hum Arduino Meter," [Online]. Available: https://www.youtube.com/watch?v=esY_OtDLv7g.

Anexos

A1 - Ficheiro main.cpp

```
#include <Arduino.h>
#include <bsec.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"
#include <Adafruit_GFX.h>
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>
#include <TFT_eSPI.h>
#include <SPI.h>
#include <DFRobot_ENS160.h>
#include <ESP32Encoder.h>
#include "Adafruit_PM25AQI.h"
#include <wifimanager.h>
#include <Adafruit_BME280.h>
#include "SparkFun_SCD4x_Arduino_Library.h"
#include <DFRobot_SGP40.h>


//DATABASE////////////////////////////////////////////
#define DEVICE "ESP32"
#define INFLUXDB_URL "https://eu-central-1-1.aws.cloud2.influxdata.com"
#define INFLUXDB_TOKEN "OvK1qsQXDixTlkudQhyGPr9STmRJH1tNfD-Ma3-
T7CqPfEi1i4mqTLZfegJ9Q-ZCp0Ef7t4j_HQUS457ThNt9A=="
#define INFLUXDB_ORG "a932d6a20e239b6a"
#define INFLUXDB_BUCKET "humidityandtemperature"
#define TZ_INFO "UTC-1"
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET,
INFLUXDB_TOKEN, InfluxDbCloud2CACert);// Declare InfluxDB client instance
with preconfigured InfluxCloud certificate
Point sensor("wifi_status");// Declare Data point
////////////////////////////////////////////////////////
//WIFI////////////////////////////////////////////////
WiFiManager wm;
////////////////////////////////////////////////////////
//BME680//////////////////////////////////////////////
/*#define SEALEVELPRESSURE_HPA (1017)
Adafruit_BME680 bme680;
float hum_weighting = 0.25; // humidity has 25% weight for IAQ score
float gas_weighting = 0.75; // gas has 75% weight for IAQ score
int   humidity_score, gas_score, air_quality_score;
float gas_reference = 2500;
float hum_reference = 40;
```

```
int   getgasreference_count = 0;
int   gas_lower_limit = 10000;  // Bad air quality limit
int   gas_upper_limit = 300000; // Good air quality limit
int   iaq;*/
//////////////////////////////////////////////////////
//BME280//////////////////////////////////////////////
Adafruit_BME280 bme280;
//////////////////////////////////////////////////////
//SCD41///////////////////////////////////////////////
SCD4x scd41;
//////////////////////////////////////////////////////
//SGP40///////////////////////////////////////////////
DFRobot_SGP40 sgp40;
//////////////////////////////////////////////////////
//LEDS////////////////////////////////////////////////
#define RED_PIN 13
#define YELLOW_PIN 12
#define GREEN_PIN 14
//////////////////////////////////////////////////////
//BUZZER//////////////////////////////////////////////
#define BUZZER_PIN 27
//////////////////////////////////////////////////////
//ENCODER/////////////////////////////////////////////
ESP32Encoder encoder;
String refresh;//need a separate frequency for encoder
unsigned long encodercurrentMillis;
unsigned long encoderpreviousMillis = 0UL;
//////////////////////////////////////////////////////
//PMS5003/////////////////////////////////////////////
#define RXD2 34 // To sensor TXD
#define TXD2 33 // To sensor RXD
int particles03um, particles05um, particles10um, particles25um,
particles50um, particles100um;
int pm10standard, pm25standard, pm100standard;
//////////////////////////////////////////////////////
//DISPLAY/////////////////////////////////////////////
TFT_eSPI tft = TFT_eSPI();
//////////////////////////////////////////////////////

//stuff to replace delays
unsigned long previousMillis = 0UL;
unsigned long interval = 10000UL;
unsigned long currentMillis;

//function declarations
void printtodisplay();
void printtoserial();
void beep(int x);
void leds();
```

```
void pushtodatabase();
String CalculateIAQ(int score);
String UpdateFrequency(int encodervalue);
void GetGasReference();
int GetHumidityScore();
int GetGasScore();
boolean readPMSdata(Stream* stream);


void setup() {
  beep(1);//one beep on startup
  Serial.begin(115200);


  tft.init(); //initialize tft display
  tft.fillScreen(TFT_BLACK);
  tft.setCursor(0, 50, 4);
  tft.setTextColor(TFT_WHITE);
  tft.fillRectVGradient(0, 0, 128, 160, TFT_ORANGE, TFT_RED);
  tft.setRotation(0); //set display rotation
  tft.println("Air\nSense");
  tft.setTextFont(2),
  tft.println("V1.0");


    // reset settings - wipe stored wifi credentials for testing
    //wm.resetSettings();

    WiFiManagerParameter system_name("system_name", "System Name",
"AirSense", 20);
    wm.addParameter(&system_name);
    wm.setTimeout(120);
    bool res;
    res = wm.autoConnect("AirSenseAP");

    if(!res) Serial.println("cant connect to wifi");
    else {
      Serial.print(system_name.getValue());
      Serial.println(" connected to wifi :)");
      beep(2);//two beeps when connected to wifi
    }

    // Connect to InfluxDB
    if (client.validateConnection()) {
      beep(3);//three beeps when connected to db
      Serial.print("Connected to InfluxDB: ");
Serial.println(client.getServerUrl());
    } else Serial.print("InfluxDB ");
Serial.println(client.getLastErrorMessage());
```

```
    // Add tags to the data sent do db
    sensor.addTag("device", DEVICE);
    sensor.addTag("System", system_name.getValue());


  Wire.begin();
  //bme680.begin(0x77);
  bme280.begin(0x76);
  scd41.begin(true, true);
  sgp40.begin();


  ESP32Encoder::useInternalWeakPullResistors=UP;
  // use pin 19 and 18 for the first encoder
  encoder.attachHalfQuad(26, 25);
  // set starting count value after attaching
  encoder.setCount(30);


  Serial1.begin(9600, SERIAL_8N1, RXD2, TXD2);//serial just for pms5003
  if (!Serial1) beep(3);



  pinMode(RED_PIN, OUTPUT);
  pinMode(YELLOW_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);//buzzer

  tft.fillScreen(TFT_BLACK);
}

struct pms5003data {
  uint16_t framelen;
  uint16_t pm10_standard, pm25_standard, pm100_standard;
  uint16_t pm10_env, pm25_env, pm100_env;
  uint16_t particles_03um, particles_05um, particles_10um,
particles_25um, particles_50um, particles_100um;
  uint16_t unused;
  uint16_t checksum;
};
struct pms5003data data;

void loop() {

  if(encoder.getCount()>60){
    encoder.setCount(60);
  }else if(encoder.getCount()<0){
```

```
    encoder.setCount(0);
  }


  if (readPMSdata(&Serial1)){
    particles03um = data.particles_03um;
    particles05um = data.particles_05um;
    particles10um = data.particles_10um;
    particles25um = data.particles_25um;
    particles50um = data.particles_50um;
    particles100um = data.particles_100um;
    pm10standard = data.pm10_standard;
    pm25standard = data.pm25_standard;
    pm100standard = data.pm100_standard;
    }


    encodercurrentMillis = millis();
    if(encodercurrentMillis - encoderpreviousMillis > 33L){
      encoderpreviousMillis = encodercurrentMillis;

        tft.setCursor(0, 145, 1);
        tft.setTextColor(TFT_WHITE,TFT_BLACK);
        tft.print(UpdateFrequency(encoder.getCount()));
    }
          while(millis()< 15000) interval = 1000UL;


    currentMillis = millis();
    if(currentMillis - previousMillis > interval){
      previousMillis = currentMillis;

      //Combine results for the final IAQ index value (0-100% where 100%
is good quality air)
      //air_quality_score = humidity_score + gas_score;
      //if ((getgasreference_count++) % 5 == 0) GetGasReference();
      //Serial.println(CalculateIAQ(air_quality_score));

      sgp40.setRhT(bme280.readHumidity(), bme280.readTemperature());

      printtodisplay();//prints sensor data to display
      printtoserial();//prints sensor data to serial
      leds();//traffic light leds
      pushtodatabase();//push sensor data to db

  }
}

void printtodisplay(){
```

```
  tft.setCursor(0, 0, 1);
  tft.setTextColor(TFT_WHITE,TFT_BLACK);
  tft.fillRectHGradient(0, 0, 160, 140, TFT_DARKGREY, TFT_DARKGREY);
  tft.setCursor(1,1);
  //tft.print("Temperature2: "); tft.print(bme280.readTemperature());
tft.println(" C");
  tft.print("Temp:");   tft.setTextSize(2);
tft.print(scd41.getTemperature()); tft.println("
C");  tft.setTextSize(1);
  tft.setCursor(1,21);
  //tft.print("Humidity2: "); tft.print(bme280.readHumidity());
tft.println(" %");
  tft.print("Hum: "); tft.setTextSize(2);tft.print(scd41.getHumidity());
tft.println(" %");tft.setTextSize(1);
  tft.setCursor(1,41);
  tft.print("CO2: "); tft.setTextSize(2);tft.print(scd41.getCO2());
tft.println(" ppm");tft.setTextSize(1);
  tft.setCursor(1,61);
  tft.print("VOC: ");
tft.setTextSize(2);tft.print(sgp40.getVoclndex());tft.println("/500");tft
.setTextSize(1);
  tft.setCursor(1,81);
  tft.print("IAQ: "); tft.setTextSize(2);tft.print("---
");tft.println("/500");tft.setTextSize(1);

  tft.setCursor(1,100);
  tft.println("PM 1.0  = " + String(pm10standard)                    );
  tft.setCursor(1,110);
  tft.println("PM 2.5  = " + String(pm25standard)                    );
  tft.setCursor(1,120);
  tft.println("PM 10.0 = " + String(pm100standard)                  );

  tft.setCursor(1,131);
  tft.print("Pressure: "); tft.print(bme280.readPressure() / 100.0);
tft.println(" hPa");

}

void printtoserial(){
  Serial.println();
  Serial.println(" Refresh Rate = " +
String(interval/1000)               + "s\n");
  //Serial.println("  Temperature = " +
String(bme680.temperature)          + "°C");
  Serial.println(" Temperature2 = " +
String(bme280.readTemperature())    + "°C");
  Serial.println(" Temperature3 = " +
String(scd41.getTemperature())      + "°C");
```

```
  //Serial.println("     Humidity = " +
String(bme680.humidity)             + "%");
  Serial.println("    Humidity2 = " +
String(bme280.readHumidity())       + "%");
  Serial.println("    Humidity3 = " +
String(scd41.getHumidity())         + "%");
  //Serial.println("     Pressure = " + String(bme680.pressure /
100)       + " hPa");
  Serial.println("    Pressure2 = " + String(bme280.readPressure() / 100)
+ " hPa");
  Serial.println("          CO2 = " +
String(scd41.getCO2())              + " ppm");
  Serial.println("    VOC Index = " +
String(sgp40.getVoclndex())              );
  Serial.println("       PM 1.0 = " +
String(pm10standard)                      );
  Serial.println("       PM 2.5 = " +
String(pm25standard)                      );
  Serial.println("      PM 10.0 = " +
String(pm100standard)               + "\n");
  Serial.println("---------------------------------------");
  Serial.print("Particles > 0.3um / 0.1L air:");
Serial.println(particles03um);
  Serial.print("Particles > 0.5um / 0.1L air:");
Serial.println(particles05um);
  Serial.print("Particles > 1.0um / 0.1L air:");
Serial.println(particles10um);
  Serial.print("Particles > 2.5um / 0.1L air:");
Serial.println(particles25um);
  Serial.print("Particles > 5.0um / 0.1L air:");
Serial.println(particles50um);
  Serial.print("Particles > 10.0 um / 0.1L air:");
Serial.println(particles100um);
}

void pushtodatabase(){
  //Clear fields for reusing the point
  sensor.clearFields();
  // Store measured value into point
  sensor.addField("temperature2", bme280.readTemperature());
  sensor.addField("humidity2", bme280.readHumidity());
  sensor.addField("pressure2", bme280.readPressure() / 100);
  sensor.addField("temperature3", scd41.getTemperature());
  sensor.addField("humidity3", scd41.getHumidity());
  sensor.addField("co2", scd41.getCO2());
  sensor.addField("vocindex", sgp40.getVoclndex());
  sensor.addField("pm10", pm10standard);
  sensor.addField("pm25", pm25standard);
  sensor.addField("pm100", pm100standard);
```

```
  sensor.addField("particles03", particles03um);
  sensor.addField("particles05", particles05um);
  sensor.addField("particles10", particles10um);
  sensor.addField("particles25", particles25um);
  sensor.addField("particles50", particles50um);
  sensor.addField("particles100", particles100um);
  //sensor.addField("airqualityscore", (100-air_quality_score) * 5);

  if (client.writePoint(sensor) && WiFi.status() == WL_CONNECTED){
    Serial.print("Writing: "); Serial.println(sensor.toLineProtocol());
  }else if(WiFi.status() == !WL_CONNECTED && millis() > 10800000 ){
    ESP.restart();
  }
}

void beep(int x){
  if(x==1){
    tone(BUZZER_PIN, 500, 200);
  }else if(x==2){
    tone(BUZZER_PIN, 500, 100);
    tone(BUZZER_PIN, 1000, 100);
  }else if(x==3){
    tone(BUZZER_PIN, 500, 75);
    tone(BUZZER_PIN, 1000, 75);
    tone(BUZZER_PIN, 1500, 75);
  }
}

void leds(){
  if(scd41.getCO2() >= 1600){
    digitalWrite(RED_PIN, HIGH); //red led
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(GREEN_PIN, LOW);
  }else if(scd41.getCO2() >= 800 && scd41.getCO2() < 1599){
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, HIGH); //yellow led
    digitalWrite(GREEN_PIN, LOW);
  }else{
    digitalWrite(RED_PIN, LOW);
    digitalWrite(YELLOW_PIN, LOW);
    digitalWrite(GREEN_PIN, HIGH); //green led
  }
}

String UpdateFrequency(int encodervalue) {
  String frequencytext = "Refresh Rate - ";
  if       (encodervalue >= 60                       ){
    interval = 300000UL;
    frequencytext += "300s  \n/                     ";
```

```
  }else if (encodervalue >= 54 && encodervalue <= 59 ){
    interval = 200000UL;
    frequencytext += "200s  \n///                  ";
  }else if (encodervalue >= 48 && encodervalue <= 53 ){
    interval = 150000UL;
    frequencytext += "150s  \n/////                ";
  }else if (encodervalue >= 42 && encodervalue <= 47 ){
    interval = 100000UL;
    frequencytext += "100s  \n///////              ";
  }else if (encodervalue >= 36 && encodervalue <= 41 ){
    interval = 80000UL;
    frequencytext += "80s   \n/////////            ";
  }else if (encodervalue >= 30 && encodervalue <= 35 ){
    interval = 60000UL;
    frequencytext += "60s   \n///////////          ";
  }else if (encodervalue >= 24 && encodervalue <= 29 ){
    interval = 50000UL;
    frequencytext += "50s   \n/////////////        ";
  }else if (encodervalue >= 18 && encodervalue <= 23 ){
    interval = 40000UL;
    frequencytext += "40s   \n///////////////      ";
  }else if (encodervalue >= 12 && encodervalue <= 17 ){
    interval = 30000UL;
    frequencytext += "30s   \n/////////////////    ";
  }else if (encodervalue >= 6  && encodervalue <= 11 ){
    interval = 20000UL;
    frequencytext += "20s   \n///////////////////  ";
  }else if (                    encodervalue <=  5 ){
    interval = 10000UL;
    frequencytext += "10s   \n////////////////////";
  }
  return frequencytext;
}

//fixes the problem with adafruit's pms library not working after 24h, no
idea how it works
boolean readPMSdata(Stream *s) {
  if (! s->available()) return false;

  // Read a byte at a time until we get to the special '0x42' start-byte
  if (s->peek() != 0x42) {
    s->read();
    return false;
  }

  // Now read all 32 bytes
  if (s->available() < 32) return false;

  uint8_t buffer[32];
```

```
  uint16_t sum = 0;
  s->readBytes(buffer, 32);

  // get checksum ready
  for (uint8_t i = 0; i < 30; i++) {
    sum += buffer[i];
  }

  // The data comes in endian'd, this solves it so it works on all
platforms
  uint16_t buffer_u16[15];
  for (uint8_t i = 0; i < 15; i++) {
    buffer_u16[i] = buffer[2 + i * 2 + 1];
    buffer_u16[i] += (buffer[2 + i * 2] << 8);
  }

  // put it into a nice struct :)
  memcpy((void *)&data, (void *)buffer_u16, 30);

  if (sum != data.checksum) {
    Serial.println("Checksum failure");
    return false;
  }
  // success!
  return true;
}
```

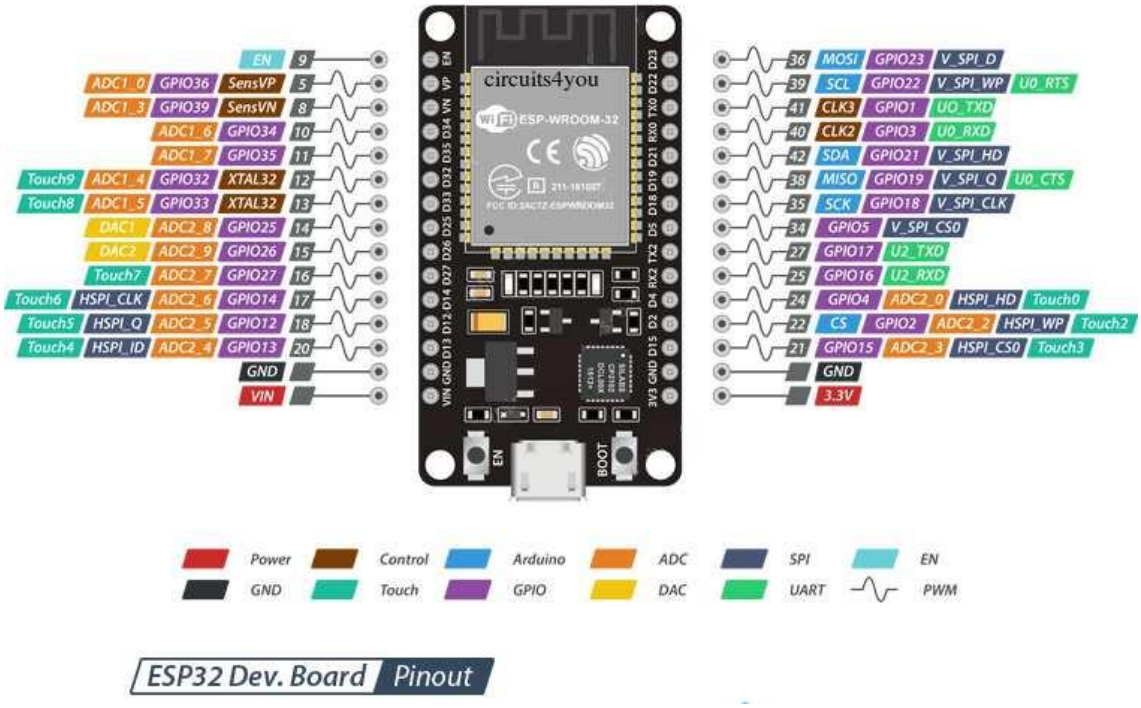
A2 - Ficheiro Platformio.ini

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
lib_deps =
    boschsensortec/BSEC Software Library@^1.8.1492
    boschsensortec/BME68x Sensor library@^1.1.40407
    adafruit/Adafruit BME680 Library@^2.0.2
    tobiasschuerg/ESP8266 Influxdb@^3.13.1
    bodmer/TFT_eSPI@^2.5.31
    adafruit/Adafruit PM25 AQI Sensor@^1.0.6
    dfrobot/DFRobot_ENS160@^1.0.1
    madhephaestus/ESP32Encoder@^0.10.1
    plerup/EspSoftwareSerial@^8.1.0
    https://github.com/tzapu/WiFiManager.git
    wifwaf/MH-Z19@^1.5.4
    adafruit/Adafruit BME280 Library@^2.2.2
    sparkfun/SparkFun SCD4x Arduino Library@^1.1.2
    dfrobot/DFRobot_SGP40@^1.0.4
monitor_speed = 115200
build_flags =
    -D USER_SETUP_LOADED=1
    -D ST7735_DRIVER=1
    -D TFT_RGB_ORDER=TFT_RGB
    -D TFT_BL=32
    -D TFT_BACKLIGHT_ON=HIGH
    -D TFT_WIDTH=128
    -D TFT_HEIGHT=160
    -D TFT_MOSI=23
    -D TFT_SCLK=18
    -D TFT_CS=15
    -D TFT_DC=2
    -D TFT_RST=4
    -D LOAD_FONT2=1
    -D LOAD_FONT4=1
    -D LOAD_FONT6=1
```

```
-D LOAD_FONT7=1
-D LOAD_FONT8=1
-D LOAD_GFXFF=1
-D SMOOTH_FONT=1
-D LOAD_GLCD=1
-D SPI_FREQUENCY=27000000
```

A3 - Diagrama de pinout ESP32 DevKit



A4 - Flux Queries

Temperatura:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "temperature"
  )
```

Humidade relativa:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "humidity"
  )
```

Pressão:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "pressure"
  )
```

CO_2:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "co2"
  )
```

VOC Index:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "vocindex"
  )
```

Particulate Matter:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "pm10"
  )
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "pm25"
  )
```

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop:v.timeRangeStop)
  |> filter(fn: (r) => r._field == "pm100"
  )
```

Maximos e mínimos diários:

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "temperature2")
  |> aggregateWindow(every: 24h, fn: max, createEmpty: false)
```

```
from(bucket: "humidityandtemperature")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_field"] == "temperature2")
  |> aggregateWindow(every: 24h, fn: min, createEmpty: false)
```