



instituto
superior de
engenharia
de lisboa

Instituto Superior de Engenharia de Lisboa

Mestrado em Engenharia Informática e de Computadores

Internet das Coisas

2º Semestre 2024/2025

Junho de 2025

Trabalho Prático – Fase 4

Sistema IoT para monitorização de qualidade de ar

Alunos:

Pedro Mateus A49329

Tiago Alves A49916

Verónica Silva A49917

Índice

Índice de Figuras	3
Índice de Tabelas.....	3
Lista de Acrónimos	4
1. Introdução	5
2. Requisitos Funcionais da Solução.....	6
3. Caracterização do Projeto.....	7
3.1. Tecnologias Utilizadas.....	7
3.1.1. Porquê?.....	7
3.1.2. Alternativas	7
3.1.3. Restrições.....	8
3.2. Caracterização do Tráfego	8
3.2.1. Frequência de envio dos dados.....	9
3.2.2. Payload e técnicas para a sua minimização	10
3.2.3. Duty Cycle.....	11
4. Arquitetura de Software	14
4.1. Dispositivo - The Things Network	16
4.2. TTN - Node-RED	17
5. Interface da Aplicação e Validação Funcional sem o Docker.....	19
5.1. Interface da Aplicação	19
5.2. Validação funcional	23
6. Interface da Aplicação e Validação Funcional com o Docker	27
6.1. Data Model e Metadata.....	27
6.2. Interface da Aplicação	29
6.3. Validação funcional	32
7. Manual de Utilização da Aplicação.....	36
7.1. Requisitos da aplicação.....	36
7.2. Funcionamento.....	37
8. Referências.....	39

Índice de Figuras

<i>Figura 3.1 - Gateways na área metropolitana de Lisboa (retirada de [4])</i>	9
<i>Figura 3.2 - Parâmetros para o cálculo do ToA no site da SemTech (disponível em [8])</i>	12
<i>Figura 4.1 - Configuração do LoRa para o nosso dispositivo</i>	16
<i>Figura 4.2 - Código para codificação e empacotamento dos dados e envio</i>	17
<i>Figura 4.3 - Código na TTN para desempacotamento dos dados</i>	17
<i>Figura 4.4 - Configuração do MQTT broker</i>	18
<i>Figura 4.5 - Arquitetura de Software geral</i>	18
<i>Figura 5.1 - Flow criado no Node-RED</i>	19
<i>Figura 5.2 - Função para processamento dos dados</i>	20
<i>Figura 5.3 - Função para colocar no dashboard os dados do ficheiro</i>	22
<i>Figura 5.4 - Função para enviar os dados para escrita</i>	22
<i>Figura 5.5 – Uplink message que chegou à TTN</i>	23
<i>Figura 5.6 - Payload do uplink_message recebido pelo Node-RED</i>	24
<i>Figura 5.7 - Conteúdo do ficheiro com o resultado da medição no ISEL na sessão anterior</i>	24
<i>Figura 5.8 - Dashboard quando iniciamos o Node-RED</i>	25
<i>Figura 5.9 - Dashboard depois da nova medição</i>	25
<i>Figura 5.10 - Ficheiro após escrita na nova sessão</i>	26
<i>Figura 5.11 - Resultado do clear data</i>	26
<i>Figura 6.1 - Modelo de dados adotado</i>	28
<i>Figura 6.2 - Flow criado no Node-RED</i>	29
<i>Figura 6.3 - Código para enviar os dados com o data model escolhido para criação das entidades</i>	30
<i>Figura 6.4 - Função para criação da subscrição</i>	31
<i>Figura 6.5 - HTTP in "Listen" para criar o URL e o Node-RED escutar as notificações</i>	31
<i>Figura 6.6 - Função implementada para recolher e representar os dados que já estavam no Orion Context Broker de sessões anteriores</i>	32
<i>Figura 6.7 - Entidades armazenadas no Orion antes da nova medição</i>	33
<i>Figura 6.8 - Dashboard quando clicamos no botão de load</i>	34
<i>Figura 6.9 - Dashboard depois da nova medição</i>	34
<i>Figura 6.10 – Conteúdo das entidades depois da nova medição</i>	35
<i>Figura 6.11 - Resultado do clear data</i>	35
<i>Figura 7.1 - Ligações para comunicação I2C com o sensor</i>	36

Índice de Tabelas

<i>Tabela 3.1 – Resultado das técnicas para minimizar o payload</i>	11
<i>Tabela 3.2 - Estudo do cumprimento do duty cycle com SF 12</i>	12
<i>Tabela 3.3 - Estudo do cumprimento do duty cycle com SF 10</i>	12
<i>Tabela 3.4 - Estudo do cumprimento do duty cycle com SF 8</i>	13
<i>Tabela 4.1 - Valor mínimo e máximo das variáveis antes da otimização</i>	14
<i>Tabela 4.2 - Valor mínimo e máximo das variáveis depois da otimização</i>	15
<i>Tabela 4.3 - Organização dos bytes</i>	15
<i>Tabela 5.1 - Dados recolhidos pelo dispositivo</i>	23
<i>Tabela 6.1 - Dados recolhidos pelo dispositivo</i>	33

Lista de Acrónimos

AQI – *Air Quality Index*

BW - *Bandwidth*

CSS - *Chirp Spread Spectrum*

eCO₂ – Dióxido de carbono equivalente

IoT – *Internet of Things*

LoRa – *Long Range*

LPWAN – *Low-power wide-area network*

MQTT - *Message Queuing Telemetry Transport*

PTx - *Transmission Power*

RSSI - *Received Signal Strength Indicator*

SF - *Spreading Factor*

SNR - *Signal-to-Noise Ratio*

TLS - *Transport Layer Security*

ToA - *Time on Air*

TVOC - *Total Volatile Organic Compounds*

VOCs - *Volatile Organic Compounds*

1. Introdução

Apesar de, na Europa, a qualidade do ar ter melhorado nas últimas décadas, os níveis de poluição continuam a ser perigosos em muitas zonas^[1] e, ainda que os efeitos da inalação de ar poluído sejam conhecidos, esta preocupação é frequentemente ignorada devido à falta de informação sobre a qualidade do ar envolvente. A pandemia da COVID-19 reforçou a importância da medição do *Air Quality Index* (AQI) em tempo real. Posto isto, diversos sistemas foram propostos para monitorizar a qualidade do ar^[2], muitos baseados na tecnologia da IoT, permitindo a obtenção de informações em tempo real sobre o ambiente.

A qualidade do ar tem um impacto direto na saúde humana e na economia. A urbanização, a desflorestação e a atividade industrial contribuem para a deterioração da qualidade do ar, tornando essencial a sensibilização da sociedade sobre esta questão. O ar poluído pode afetar gravemente a saúde, especialmente em áreas urbanas onde as pessoas passam grande parte do tempo ao ar livre. A monitorização da qualidade do ar permite identificar a presença de substâncias nocivas e adotar medidas preventivas.

Os sistemas de monitorização da qualidade do ar têm evoluído significativamente com os avanços da tecnologia IoT. Estes utilizam dispositivos inteligentes interligados para medir em tempo real os níveis de poluição e outros parâmetros como temperatura e umidade. As tecnologias *Low-Power Wide-Area Network* (LPWAN), em particular a *Long Range* (LoRa), destacam-se por permitirem uma comunicação eficiente, de baixo consumo energético e de longa distância, ideal para aplicações ambientais. Isso permite a transmissão de pequenas quantidades de dados a grandes distâncias^[3].

Deste modo, este projeto tem como objetivo desenvolver e implementar uma solução baseada em IoT para medir a qualidade do ar em tempo real e apresentar os resultados de forma rápida e o mais precisa possível, assim como efetuar o envio dos dados para a *cloud* e realizar o devido processamento dos mesmos.

Nota: Foi criado um Github com os ficheiros docker-compose.yml para correr a base de dados mongoDB, o context broker Orion e o Node-red, datamodel.json para demonstrar a estrutura de dados que está no Orion, nodered.json que pode ser importando para ser apresentado o flow node-red implementado, e os ficheiros boot_codigo_final.py e main_codigo_final.py que são os ficheiros boot.py e main.py finais na TTGo. O github pode ser acedido através deste link: <https://github.com/PedroMateus09/IoT.git> .

2. Requisitos Funcionais da Solução

O objetivo principal deste projeto é o desenvolvimento de uma solução IoT para monitorização da qualidade do ar, utilizando o TTGO T-Beam, juntamente com o sensor CCS811. A solução será capaz de medir a concentração de *Total Volatile Organic Compounds* (TVOC), que é um grupo de *Volatile Organic Compounds* (VOCs), e que são utilizados para representar todo o conjunto de poluentes, assim como será capaz de medir a concentração de dióxido de carbono equivalente (eCO_2) no ambiente, apresentando os dados num *dashboard*.

Os dados serão recolhidos pelo sensor CCS811, que envia os mesmos para o TTGO T-Beam através de comunicação I2C. Assim que este recebe os dados, os mesmos serão processados para serem enviados para uma *gateway*.

Para além destes dados de qualidade de ar enviados para o microcontrolador, irá ainda ser utilizado o GPS integrado neste, para saber o ponto geográfico onde os dados foram recolhidos e assim podermos associar os dados a uma localização.

Tal como referido anteriormente, os dados serão apresentados numa interface gráfica, mostrando assim:

- Um mapa onde será possível ver os dados georreferenciados previamente recolhidos.
- Uma tabela com os pontos geográficos e os respetivos resultados.
- Um gráfico com a percentagem de localizações com boa qualidade de ar.

Existirá ainda a possibilidade de aceder a dados específicos e detalhados de uma localização através do mapa, clicando no ponto do mapa que queremos ver os resultados.

Vale ainda realçar que o microcontrolador irá ser alimentado por uma bateria, de maneira que possa ser utilizado sem uma fonte de alimentação fixa. Assim sendo, com um sistema de baixo consumo de energia, o dispositivo pode operar de maneira contínua por longos períodos com uma única carga, o que o torna ideal para aplicações de monitorização ambiental em áreas remotas ou em locais de difícil acesso.

Este sistema permitirá então recolher dados de forma dinâmica em vários pontos geográficos, com um baixo consumo de energia e com um baixo custo comparado com outras soluções.

3. Caracterização do Projeto

3.1. Tecnologias Utilizadas

Neste projeto apenas iremos precisar de utilizar uma tecnologia de comunicação, sendo esta o LoRa.

LoRa é uma técnica de modulação sem fios derivada da tecnologia *Chirp Spread Spectrum* (CSS). Esta codifica informações em ondas de rádio utilizando pulsos *chirp*, que são sinais em que a frequência varia com o tempo [3].

3.1.1. Porquê?

A escolha desta tecnologia prende-se por diversos fatores importantes num projeto desta natureza, sendo alguns destes:

- O LoRa é ideal para aplicações que transmitem pequenos blocos de dados com baixas taxas de bits, como é o caso da nossa aplicação;
- Longo alcance, uma vez que os dados podem ser transmitidos a uma distância maior em comparação com tecnologias como Wi-Fi, Bluetooth ou ZigBee;
- Baixo consumo de energia, o que é essencial para sistemas alimentados a bateria;
- Infraestrutura flexível, com suporte para redes privadas e públicas;
- Espectro livre de licença.

3.1.2. Alternativas

Existem algumas alternativas ao LoRa, que são também tecnologias LPWAN, e que apresentam também os seus pontos fortes e fracos, como se pode ver na figura 3.1.

Tecnologia	Taxa de transmissão de dados	Consumo	Comentários/Comparação com LoRa
Wi-Fi	Alto ($\approx 100\text{Mbps}$)	Alto	Inadequado para áreas remotas e consome mais energia
SigFox	Baixo ($\approx 1\text{kbps}$)	Baixo	<i>Subscription-based model</i>
NB-IoT	Médio ($\approx 100\text{kbps}$)	Médio	Espetro licenciado, o que aumenta o custo
LTE-M	Médio ($\approx 100\text{kbps}$)	Médio	

Existem ainda outras tecnologias de curto alcance, como o BLE, o NFC e o Zigbee, que não foram incluídas na tabela por não se adequarem às necessidades do nosso projeto, uma vez que este requer comunicações de longo alcance.

Após a análise das várias alternativas, concluiu-se que a tecnologia LoRa é a mais adequada para este projeto, pois oferece um alcance elevado, baixo consumo energético, uma grande capacidade de rede, comunicação robusta, baixo custo e é especialmente eficiente na transmissão de pequenos volumes de dados — características ideais para a nossa aplicação, dado que os dados recolhidos não têm grande dimensão. Adicionalmente, o TTGO T-beam já integra um módulo LoRa, permitindo fácil implementação.

3.1.3. Restrições

Embora a tecnologia LoRa apresente vantagens significativas, é importante considerar as suas limitações e restrições, como:

- Baixa taxa de transferência de dados, que é limitada a 0,3-50kbps, o que é um valor significativamente inferior a outras tecnologias;
- Limitações do *Duty Cycle*, que na europa é de apenas 1%, o que dá um total de 36 segundos por hora;
- Como apresenta um maior alcance, apresenta também maior latência;
- Necessidade de uma *gateway*, apesar de já existirem diversas para utilização publica;

No caso deste projeto, algumas restrições podem ter impactos significativos como:

- Frequência limitada de envio de dados, que pode levar à necessidade de otimizar os intervalos de recolha de dados e transmissão;
- Limitação na quantidade de dados transmitidos, que irá ser minimizada uma vez que irá existir um processamento local dos dados antes da transmissão;

3.2. Caracterização do Tráfego

Neste capítulo irá ser estudada e apresentada a caracterização do tráfego, isto é, a frequência com que os dados irão ser enviados, o tamanho do *payload* que irá ser enviado, assim como técnicas para o minimizar e, por fim, calcular o *duty cycle* utilizado e assegurar o seu cumprimento.

Vale ainda realçar que o sistema proposto vai ser desenvolvido para funcionar de forma móvel, recolhendo dados da qualidade do ar em vários pontos da área metropolitana de Lisboa. O objetivo é cobrir o máximo possível desta área ao longo do tempo.

Como o sistema depende de *gateways* públicas, o requisito de cobertura de comunicação é a presença de pelo menos uma *gateway* disponível na proximidade do local de envio. Ou seja, define-se como requisito de cobertura mínimo a existência de ligação com uma *gateway* a uma distância máxima de cerca de 1 a 2 km num ambiente urbano.

Este modelo de cobertura dinâmico faz com que não seja necessário um planeamento fixo de infraestrutura, mas exige uma disponibilidade de *gateways* na área urbana de Lisboa, o que não deverá ser um problema dado que esta zona dispõe de uma boa cobertura de gateways da rede pública, viabilizando desta forma esta estratégia.

Abaixo encontra-se uma figura que ilustra a área abrangida e as *gateways* disponíveis em Lisboa, informação esta que pode ser encontrada no site da *The Things Network*. São um total de 17 *gateways* e 70 colaboradores.

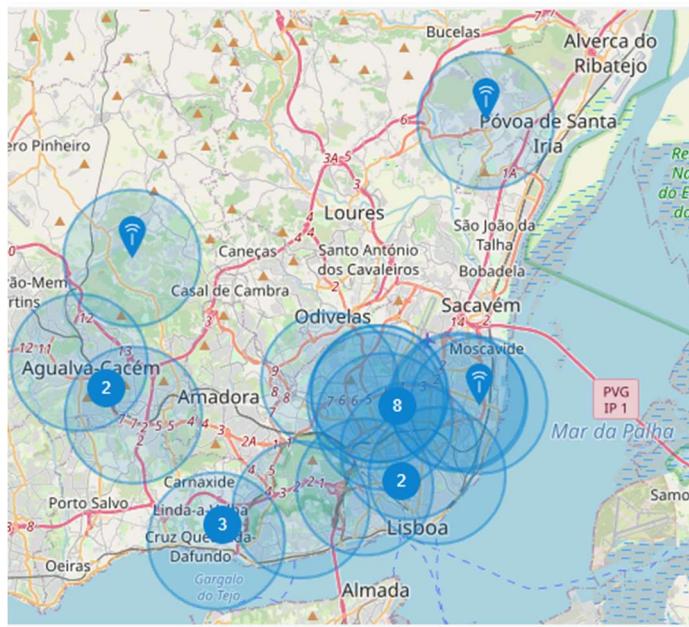


Figura 3.1 - Gateways na área metropolitana de Lisboa (retirada de [4])

3.2.1. Frequência de envio dos dados

Dado que o sensor que iremos utilizar necessita de vinte minutos de utilização antes dos valores se poderem considerar válidos, durante esse tempo iremos apenas ler os valores, sem os processar nem enviar.

O tempo de espera entre a leitura de valores do sensor irá ser de dez segundos, somando assim cento e vinte valores antes de termos valores válidos, uma vez que $10 \times 120 = 1200$ segundos, que equivale a vinte minutos.

O módulo NEO-6M GPS também necessita de um tempo para os valores começarem a ser válidos, então os vinte minutos que esperamos pelo sensor de qualidade de ar servem perfeitamente para esperar por este sensor também.

Após passarem os vinte minutos, iremos passar cinco minutos a ler os valores do sensor de qualidade de ar, armazenando-os. Após passarem os cinco minutos, que correspondem a trinta valores armazenados, irá ser feita a média do eCO2 e do TVOC desses trinta valores, e apenas irá ser enviado por LoRa essa média, juntamente com os valores de longitude e latitude provenientes do sensor GPS. Este valor geográfico idealmente irá apenas ser lido uma vez, dado que apenas vamos atribuir a média dos valores de qualidade de ar a uma única localização.

Em suma, após o tempo de espera pelos sensores, os valores serão lidos e armazenados durante cinco minutos, processando-os posteriormente e enviando por LoRa. Sendo assim, a frequência de envio dos dados irá ser de cinco minutos. Se enviarmos durante 1 hora, isso significa um total de 12 mensagens por hora.

3.2.2. Payload e técnicas para a sua minimização

A mensagem que irá ser enviada por LoRa com os dados lidos dos sensores após processamento irá ter a seguinte estrutura:

```
lat:<Latitude> long:<Longitude> co2:<Valor eCO2> tvoc:<Valor TVOC>
```

Uma técnica que poderíamos adotar neste caso seria encurtar os nomes dos “rótulos” e juntar a latitude com a longitude, ficando na seguinte forma:

```
lt,lg:<Latitude>,<Longitude> c:<Valor eCO2> t:<Valor TVOC>
```

Se for necessário minimizar ainda mais o tamanho do payload podemos utilizar a seguinte estrutura, sem os rótulos:

```
<Latitude>,<Longitude>|<Valor eCO2>|<Valor TVOC>
```

Se ainda assim esta redução não for suficiente, poderíamos utilizar uma codificação para binário de forma a serem utilizados menos bytes, utilizando para isto dois inteiros com sinal de 32 bit para representar a latitude e longitude, e dois inteiros sem sinal de 16 bit para representar o eCO2 e o TVOC, ficando da seguinte maneira:

```
[lat:int32][lon:int32][eCO2:uint16][tvoc:uint16]
```

Para uma estrutura mais legível e que fosse de fácil interpretação seria melhor utilizar a primeira abordagem, mas se o tamanho do *payload* necessitar de ser encurtado de maneira a cumprir o *duty cycle* será necessário utilizar uma das outras estruturas. Será por isso necessário realizar os cálculos, posteriormente, para determinar a melhor abordagem.

Tabela 3.1 – Resultado das técnicas para minimizar o payload

Estrutura da mensagem	Exemplo	<i>Payload [Byte]</i>
Menos compacta	lat:38.7169 long:-9.1399 co2:415 tvoc:128	41
Rótulos menores	lt,lg:38.7169,-9.1399 c:415 t:128	33
Mais compacta	38.7169,-9.1399 415 128	23
Binário	003b1fea ffbeb67a 019f 0080	12

Para reduzir ainda mais o *payload*, dependendo das consequências para os resultados, podíamos ainda reduzir a precisão geográfica, passando a latitude e longitude de inteiro de 32 bit com sinal para 16 bit ^[5].

3.2.3. Duty Cycle

O *duty cycle* indica a fração de tempo que um recurso fica ocupado e este é frequentemente regulamentado pelo governo.

Na Europa, os ciclos de trabalho são regulamentados pela seção 4.3.3 da norma ETSI EN300.220-2 V3.2.1 (2018-06) ^[6], que define um limite de 1% para a faixa ISM que vamos utilizar, que é de 868MHz ^[7].

Isso significa que, num intervalo de 100 segundos, um dispositivo pode transmitir por, no máximo, 1 segundo. Portanto, sendo o nosso intervalo de 300 segundos (5 minutos), o dispositivo pode transmitir por, no máximo, 3 segundos.

Para garantir conformidade com o *duty cycle*, é essencial calcular o ToA, e verificar se, com esse tempo e essa frequência de envio, não se ultrapassa o limite imposto.

O ToA depende de vários parâmetros de configuração LoRa, como o SF, a BW, o *coding rate*, a presença de cabeçalhos e o tamanho do *payload*.

Iremos então calcular o ToA para as várias estruturas de *payload* discutidas na secção anterior, utilizando um BW de 125kHz, um CR de 4/5 e três diferentes valores de SF, sendo eles 8, 10 e 12. Para realizar este cálculo iremos utilizar a calculadora da Semtech ^[8], com os campos preenchidos como ilustrado na figura abaixo.

Iremos também calcular o *duty cycle* para cada um deles utilizando a fórmula:

$$\frac{\text{ToA}}{\text{Total time(5 minutes)}} \times 100.$$

The figure shows a user interface for configuring LoRaWAN parameters. It consists of five main sections: **Device**, **RF**, **Modem**, **Packet**, and **Protocol**. Each section contains dropdown menus and input fields for setting specific parameters. A green **SUBMIT >** button is located at the bottom center.

Device	RF	Modem	Packet	Protocol
Device: SX127X RF Path: RF Switch	Tx Power: 10 dBm Receiver Mode: High Sensitivity PA Boost: Off Ramp Time: 3400 us Frequency: 868000000 Hz	Modulation: LoRa Spreading Factor: 12 Bandwidth: 125 kHz Coding Rate: 4/5 Low Data Rate Optimizer: ON	Preamble Length: 8 Symbols Header: Enabled Payload Length: Bytes CRC: ON	TX Period: 400 ms RX Duration: 400 ms RX Period: 400 ms Sleep Consumption: 5 μA

Figura 3.2 - Parâmetros para o cálculo do ToA no site da SemTech (disponível em [8])

Tabela 3.2 - Estudo do cumprimento do duty cycle com SF 12

SPREADING FACTOR = 12

Estrutura da mensagem	<i>Payload</i> [Byte]	ToA [ms]	<i>Duty Cycle (%)</i>
Menos compacta	41	2140	0.7133
Rótulos menores	33	1810	0.6033
Mais compacta	23	1480	0.4933
Binário	12	1160	0.3867

Tabela 3.3 - Estudo do cumprimento do duty cycle com SF 10

SPREADING FACTOR = 10

Estrutura da mensagem	<i>Payload</i> [Byte]	ToA [ms]	<i>Duty Cycle (%)</i>
Menos compacta	41	616.45	0.2055
Rótulos menores	33	534.53	0.1782
Mais compacta	23	411.65	0.1372
Binário	12	329.73	0.11

Tabela 3.4 - Estudo do cumprimento do duty cycle com SF 8

SPREADING FACTOR = 8

Estrutura da mensagem	<i>Payload</i> [Byte]	ToA [ms]	<i>Duty Cycle (%)</i>
Menos compacta	41	195.07	0.065
Rótulos menores	33	164.35	0.055
Mais compacta	23	133.63	0.0445
Binário	12	92.67	0.031

Como podemos observar nas tabelas de resultados, em todos as estruturas de *payload*, desde a mais compacta à mais descritiva, para todos os SF, o *duty cycle* de 1% é cumprido, tendo em conta o intervalo de 5 minutos entre transmissões.

4. Arquitetura de Software

Após estudar todas as soluções para otimização do tamanho do *payload* apresentadas na secção 3 deste documento, tentámos arranjar uma forma ainda mais otimizada de enviar os valores por LoRaWAN para uma *gateway* da TTN. A solução desenvolvida utiliza apenas 7 byte para enviar os dados de qualidade de ar (eCO2 e TVOC) e os dados de localização geográfica (latitude e longitude). Abaixo irá ser detalhada e explicada essa solução.

Antes de abordarmos a organização dos dados em cada byte, é importante perceber os valores possíveis de cada um dos dados, para sabermos quantos bits seriam necessários para englobar todos os valores. Posto isto, na tabela 4.1 é apresentada essa informação.

Tabela 4.1 - Valor mínimo e máximo das variáveis antes da otimização

Variável	Valor mínimo	Valor máximo
TVOC	0	1187
eCO2	400	8192
Latitude	-90,000000	90,000000
Longitude	-180,000000	180,000000

Se observarmos os valores do TVOC e do eCO2 repararmos que iria ser necessário um grande número de bits, quando na verdade podemos reduzir esse número ao diminuir a precisão que, para esta aplicação, não é necessária, uma vez que, para classificar a qualidade do ar, não observamos os valores à unidade, mas sim à centena, ou seja, não precisamos de saber que um valor é, por exemplo, 527, uma vez que saber que é 500 é suficiente para aferir a qualidade do ar.

Posto isto, na nossa aplicação, para enviar por LoRaWAN para a *gateway* iremos dividir o valor do eCO2 e do TVOC por 100 e, posteriormente, quando chegar à TTN, o valor é multiplicado por 100 de maneira a, por exemplo, o valor 527 corresponder ao valor 500.

Quanto à latitude e à longitude, antes de enviar para a TTN vamos multiplicar o valor por 10000, de maneira a preservar 4 casas decimais, assegurando ainda uma precisão aceitável. Após essa multiplicação iremos somar à latitude 900.000 e à longitude 1.800.000, com o objetivo de trabalharmos apenas com valores com sinal positivo.

Desta forma, na tabela 4.2 são apresentados os valores mínimos e máximos otimizados para cada variável, assim como o número de bits necessário para os representar.

Tabela 4.2 - Valor mínimo e máximo das variáveis depois da otimização

Variável	Valor mínimo	Valor máximo	Número de bits necessários
TVOC	0	11	4
eCO2	4	81	7
Latitude	0	1.800.000	21
Longitude	0	3.600.000	22

Posto isto, o número de bits necessários para empacotar todos os valores será de 54, o que corresponde a serem necessários 7 byte, sobrando 2 bits que irão ser ignorados.

A organização de cada byte vai ser a demonstrada na tabela 4.3:

Tabela 4.3 - Organização dos bytes

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	eCO2 (bit 6)	eCO2 (bit 5)	eCO2 (bit 4)	eCO2 (bit 3)	eCO2 (bit 2)	eCO2 (bit 1)	eCO2 (bit 0)	TVOC (bit 3)
1	TVOC (bit 2)	TVOC (bit 1)	TVOC (bit 0)	Latitude (bit 20)	Latitude (bit 19)	Latitude (bit 18)	Latitude (bit 17)	Latitude (bit 16)
2	Latitude (bit 15)	Latitude (bit 14)	Latitude (bit 13)	Latitude (bit 12)	Latitude (bit 11)	Latitude (bit 10)	Latitude (bit 9)	Latitude (bit 8)
3	Latitude (bit 7)	Latitude (bit 6)	Latitude (bit 5)	Latitude (bit 4)	Latitude (bit 3)	Latitude (bit 2)	Latitude (bit 1)	Latitude (bit 0)
4	X	X	Longitude (bit 21)	Longitude (bit 20)	Longitude (bit 19)	Longitude (bit 18)	Longitude (bit 17)	Longitude (bit 16)
5	Longitude (bit 15)	Longitude (bit 14)	Longitude (bit 13)	Longitude (bit 12)	Longitude (bit 11)	Longitude (bit 10)	Longitude (bit 9)	Longitude (bit 8)
6	Longitude (bit 7)	Longitude (bit 6)	Longitude (bit 5)	Longitude (bit 4)	Longitude (bit 3)	Longitude (bit 2)	Longitude (bit 1)	Longitude (bit 0)

Neste projeto, os dados são recolhidos por um dispositivo LoRaWAN e enviados para um *gateway* LoRaWAN, que os transmite para a *The Things Network* (TTN), onde são descodificados. A partir do TTN, os dados são encaminhados via MQTT para o Node-RED, onde podem ser visualizados num *dashboard*.

4.1. Dispositivo - The Things Network

O TTGO utiliza LoRaWAN para comunicar com *gateways* LoRa ligados à internet, que por sua vez reencaminham os dados para a TTN. Para que esta comunicação funcione corretamente, é necessário a configuração do dispositivo com os dados de autenticação através do método ABP. No código, devem ser definidos os valores de *dev_addr*, *nwk_swkey* e *app_swkey*, que podem ser acedidos na TTN, no dispositivo criado, e a frequência correspondente à região que neste caso é LoRa.EU868 e, posteriormente, realizar o *join*, tal como podemos ver na figura 4.1.

Nota: O valor das *keys* e do *addr* foi desfocado uma vez que é o valor atribuído ao nosso dispositivo. Estes valores podem ser acedidos em TTN->Device->Overview

```
lora = LoRa(mode=LoRa.LORAWAN, region=LoRa.EU868)

dev_addr = [REDACTED]
nwk_swkey = ubinascii.unhexlify('[REDACTED]')
app_swkey = ubinascii.unhexlify('[REDACTED]')

# join a network using ABP (Activation By Personalization)
lora.join(activation=LoRa.ABP, auth=(dev_addr, nwk_swkey, app_swkey))

# create a LoRa socket
sock = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# set the LoRaWAN data rate
sock.setsockopt(socket.SOL_LORA, socket.SO_DR, 0)

# make the socket blocking
# (waits for the data to be sent and for the 2 receive windows to expire)
sock.setblocking(True)
```

Figura 4.1 - Configuração do LoRa para o nosso dispositivo

Após configurarmos o LoRa, tivemos de empacotar os dados da maneira que apresentámos anteriormente, em 7 byte. O código para o empacotamento e envio via LoRaWAN é o apresentado na figura 4.2.

```

eco2 = (int)(s.eco2/100)
tvoc = (int)(s.tvoc/100)
lat = (int)(lat_decimal * 10000) + 900000
lon = (int)(lon_decimal * 10000) + 1800000
payload = []
payload.append((eco2 << 1) | ((tvoc >> 3) & 0x01))
payload.append(((tvoc & 0x07) << 5) | ((lat >> 16) & 0x1F))
payload.append((lat >> 8) & 0xFF)
payload.append(lat & 0xFF)
payload.append((lon >> 16) & 0x3F)
payload.append((lon >> 8) & 0xFF)
payload.append(lon & 0xFF)
#print("Payload:", [hex(b) for b in payload])
sock.send(bytes(payload))

print("sent")
time.sleep(5)

```

Figura 4.2 - Código para codificação e empacotamento dos dados e envio

4.2. TTN - Node-RED

Já do lado da TTN configurámos o *payload formatter* na direção *uplink* para que, quando a TTN receber os dados, estes sejam desempacotados e colocados no seu valor real, para que quando se envie por MQTT os dados estejam já dessa forma mais processada, não sendo por isso necessário desempacotar no Node-RED. O código tem como linguagem o JavaScript e é apresentado na figura 4.3.

```

Formatter code*
1 function decodeUplink(input) {
2     let bytes = input.bytes;
3     let eco2 = (((bytes[0] >> 1) & 0x7F) * 100;
4     let tvoc = (((bytes[0] & 0x01) << 3) | (bytes[1] >> 5)) & 0x0F) * 100;
5     let lat = (((bytes[1] & 0x1F) << 16) | ((bytes[2] & 0xFF) << 8) | (bytes[3] & 0xFF)) - 900000) / 10000;
6     let long = (((bytes[4] & 0x3F) << 16) | ((bytes[5] & 0xFF) << 8) | (bytes[6] & 0xFF)) - 1800000) / 10000;
7     return {
8         data: {
9             eco2,
10            tvoc,
11            lat,
12            long
13        },
14        warnings: [],
15        errors: []
16    };
17}
18

```

Figura 4.3 - Código na TTN para desempacotamento dos dados

Após receber e desempacotar os dados, a TTN reencaminha essa informação para outras plataformas através de integrações, sendo o MQTT uma das mais utilizadas. Para configurar esta integração, basta aceder ao TTN Console, entrar na aplicação criada e ativar a opção de integração via MQTT. A TTN fornece os dados de ligação, que são o broker, o porto, o nome de utilizador e uma API Key com permissões de leitura que funciona como palavra-passe.

Os dados são então publicados em MQTT com uma estrutura específica e são enviados no formato JSON.

Do lado do Node-RED, o fluxo começa com a configuração de um cliente MQTT que se liga à plataforma TTN para receber os dados enviados pelos dispositivos LoRaWAN. Para isso, adiciona-se um nó MQTT IN, onde se define o broker, o porto e as credenciais fornecidas pela TTN.

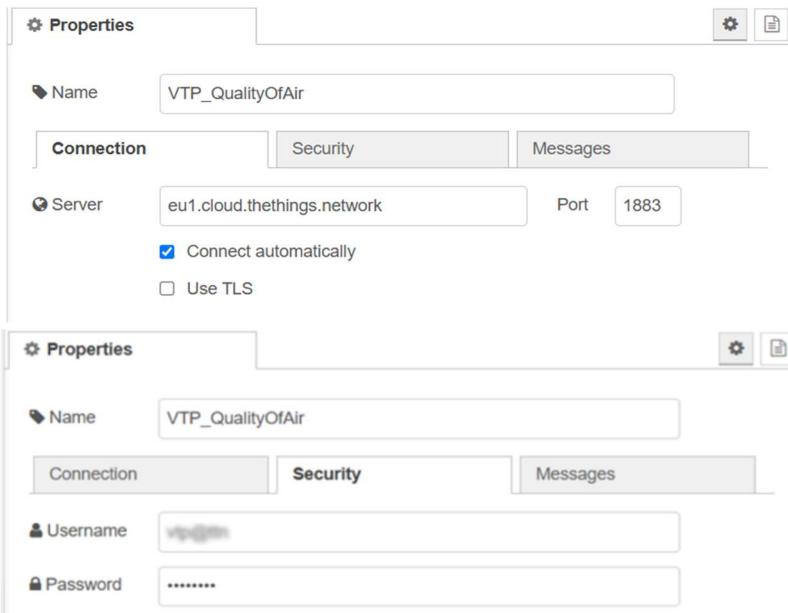


Figura 4.4 - Configuração do MQTT broker

Nota: Estas credenciais para conexão podem ser obtidas na TTN na secção “*Other Integrations -> MQTT*”

Posto isto, a arquitetura de software geral da aplicação é ilustrada na figura 4.5.

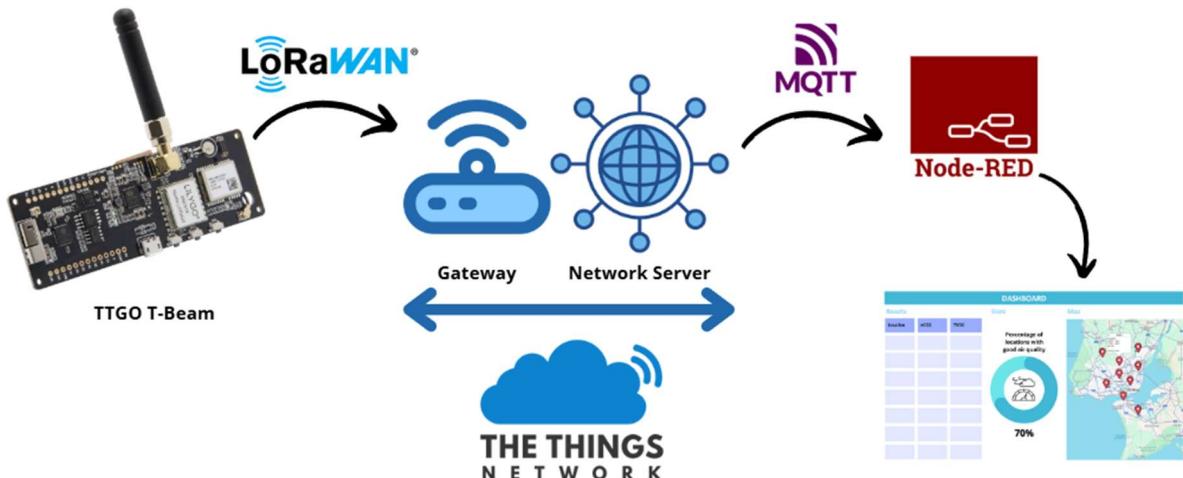


Figura 4.5 - Arquitetura de Software geral

5. Interface da Aplicação e Validação Funcional sem o Docker

5.1. Interface da Aplicação

Após todas estas configurações foi necessário criar um *flow* no Node-RED com o objetivo de apresentar estes dados recebidos, que pode ser observado na figura 5.1.

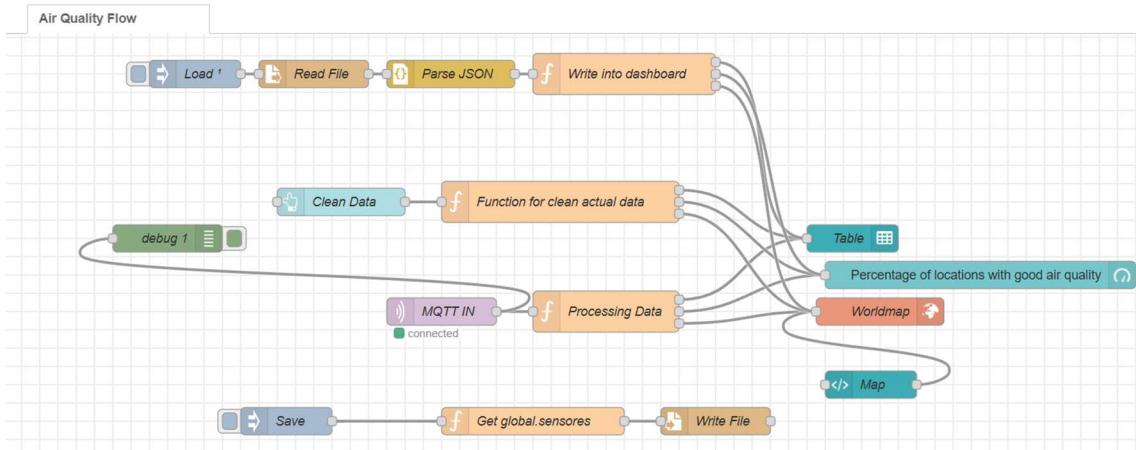


Figura 5.1 - Flow criado no Node-RED

Com o objetivo de simplificar a explicação, podemos dividir este *flow* em 3 subconjuntos, sendo estes:

- O conjunto de nós principal, que começa no MQTT IN e termina nos componentes que vão constituir o *dashboard*, sendo estes a *table*, o *gauge* e o *map*. Este contém ainda um botão para limpar os dados. Este *flow* pode ser observado como se fosse apenas constituído pelos nós do meio da figura 5.1;
- O conjunto de nós para leitura dos dados de um ficheiro que armazena os dados já recolhidos. Este pode ser observado como se fosse apenas constituído pelos nós de cima da figura 5.1;
- O conjunto de nós para escrita dos dados num ficheiro para armazenamento. Este pode ser observado como se fosse apenas constituído pelos nós de baixo da figura 5.1.

Começando pelo conjunto de nós principal, o primeiro nó é um MQTT in que, como vimos anteriormente irá receber os dados provenientes da TTN. Após este nó foi colocado uma função denominada “*processing data*” onde os dados são processados, divididos e enviados para o respetivo output: a tabela, o gráfico e o mapa. Desta maneira, esta função necessita de ter três outputs.

Assim sendo, o código JavaScript apresentado na figura 5.2 processa os dados ambientais e de GPS recebidos via MQTT do microcontrolador TTGO, extraindo a latitude, longitude, eCO2 e TVOC do *payload*, que já estão com os valores reais, avaliando ainda a qualidade do ar. Isto é, se o valor de eCO2 for igual ou inferior a 800 ppm e o de TVOC igual ou inferior a 200 ppb, a qualidade é considerada "Boa", caso contrário é considerada "Má" (ver nota abaixo). De seguida, cria um objeto novo com essas informações, juntamente com um identificador único, o nome do dispositivo e a hora de medição, adicionando-o posteriormente a uma lista global denominada *locations* para persistência entre execuções. Por fim, o código gera três saídas: uma tabela com os dados e localização, a percentagem de localizações com boa qualidade do ar (para o gráfico), e um marcador geográfico personalizado para visualização no mapa.

Nota: De acordo com as tabelas demonstradas no primeiro relatório deste projeto, podemos dizer que locais com um valor de eCO2 menor que 800 e TVOC menor que 200 apresentam uma boa qualidade de ar.

```

1 const dados = msg.payload.uplink_message.decoded_payload;
2 const lat = dados.lat;
3 const lon = dados.long;
4 const eco2 = dados.eco2;
5 const tvoc = dados.tvoc;
6
7 const goodQuality = eco2 <= 800 && tvoc <= 200;
8 const quality = goodQuality ? "Good" : "Bad";
9
10 let locations = global.get('locations') || [];
11
12 const newPoint = {
13   id: Date.now(),
14   device: `Sensor TTGO`,
15   lat: lat.toFixed(4),
16   lon: lon.toFixed(4),
17   eco2: eco2,
18   tvoc: tvoc,
19   quality: quality,
20   timestamp: new Date().toLocaleTimeString()
21 };
22
23
24
25
26
27 return [
28   // Output 1: table
29   {
30     payload: locations.map(loc => ({
31       "localização (lat, long)": `${loc.lat}, ${loc.lon}`,
32       "eCO2 (ppm)": loc.eco2,
33       "TVOC (ppb)": loc.tvoc,
34     }))
35   },
36   // Output 2: gauge
37   {
38     payload: Math.round(
39       (locations.filter(l => l.quality === "Good").length / locations.length) * 100
40     )
41   },
42   // Output 3: map
43   {
44     payload: [
45       {
46         lat: parseFloat(newPoint.lat),
47         lon: parseFloat(newPoint.lon),
48         name: `${newPoint.device}_${newPoint.id}`,
49         icon: "circle",
50         iconColor: newPoint.quality === "Good" ? "green" : "red",
51         popup: `Latitude: ${newPoint.lat} <br>Longitude: ${newPoint.lon}
52             <br><b>Environmental Data:</b><br>eCO2: ${l.eco2} ppm<br>
53             TVOC: ${l.tvoc} ppb<br>Quality of Air: ${l.quality}`,
54         layer: "locations"
55       }
56     ]
57   }
58 ];

```

Figura 5.2 - Função para processamento dos dados

Os restantes nós deste conjunto principal, foram configurados da seguinte maneira:

- Na tabela foram colocadas 3 colunas para receber a localização (lat, long), o eCO2 e o TVOC;
- O gráfico foi configurado para ser um gráfico em tarte, em que a percentagem será a percentagem de localizações com boa qualidade de ar, ficando vermelho se a percentagem for baixa, amarelo se for média e verde se for alta.

- O mapa foi configurado com uma latitude, longitude e zoom inicial para se focar na área de lisboa desde que o mapa é criado.
- Um *template* para colocar o mapa no *dashboard*, uma vez que o mapa não faz parte da mesma paleta de nós dos outros dois, então ficaria num domínio à parte, que seria /worldmap. Posto isto, colocou-se um *template* com essa *source* (/worldmap) e assim o mapa fica no mesmo domínio dos restantes componentes e, consequentemente, no *dashboard*. Assim sendo, se quisermos aceder apenas ao mapa sem o restante *dashboard* podemos aceder ao domínio /worldmap e será apenas apresentado o mapa em maior plano.
- Um botão para limpar os dados do *dashboard*, que quando é pressionado, executa uma função que coloca a tabela como vazia, o gráfico a zero e elimina os pontos do mapa.

Passando para o conjunto de nós que possibilita a leitura de um ficheiro que armazena os dados recolhidos, este começa por um nó *inject* denominado “*load*” que foi configurado para executar apenas uma vez assim que se iniciar o Node-RED. Ligou-se esse nó a um nó de leitura de ficheiro, configurando o mesmo com o caminho para o ficheiro. Ligou-se então esse nó a um nó de “*parse JSON*”, para que, na função que irá ficar ligada a este nó, consigamos aceder ao valor das variáveis de maneira mais simplificada, não precisando de código adicional para a conversão.

Por fim, na função que irá processar os dados que foram lidos dos ficheiros, começamos por colocar os dados na variável global “*locations*”, uma vez que os dados são guardados no ficheiro com a mesma estrutura e campos que é composta a variável global e, portanto, fica mais simples de colocar os novos dados nessa variável. Após isto, resta apenas preparar os dados para os 3 outputs, sendo estes a tabela, o gráfico e mapa, à semelhança da função apresentada na figura 5.2. Esta função é demonstrada na figura 5.3.

```

1  const locations = msg.payload;
2
3  global.set("locations", locations);
4
5  // Output 1: table
6  const table = locations.map(l => ({
7      "localização (lat, long)": `${l.lat}, ${l.lon}`,
8      "eCO2 (ppm)": l.eco2,
9      "TVOC (ppb)": l.tvoc
10 }));
11
12 // Output 2: gauge
13 const perGoodQuality = Math.round(
14     (locations.filter(l => l.quality === "Good").length / locations.length) * 100
15 );
16
17 // Output 3: map
18 const map = locations.map(l => ({
19     payload: {
20         lat: parseFloat(l.lat),
21         lon: parseFloat(l.lon),
22         name: `${l.device}_${l.id}`,
23         icon: "circle",
24         iconColor: l.quality === "Good" ? "green" : "red",
25         popup: `Latitude: ${l.lat} <br>Longitude: ${l.lon} <br>
26             <b>Environmental Data:</b><br>eCO2: ${l.eco2} ppm<br>
27             TVOC: ${l.tvoc} ppb<br>Quality of Air: ${l.quality}`,
28         layer: "locations"
29     }
30 }));
31
32 return [ { payload: table }, { payload: perGoodQuality }, map ];

```

Figura 5.3 - Função para colocar no dashboard os dados do ficheiro

Por último lugar, falta apenas falar do conjunto de nós que processa a escrita no ficheiro para armazenamento dos dados. Este começa com um nó *inject* mas, ao contrário do outro que injetava uma vez quando iniciava o Node-RED e depois não injetava mais, quando quisermos escrever na file apenas temos de clicar no botão do nó para que a função que está ligada a este execute. O que esta função faz é apenas aceder à variável global, converter essa variável para uma string JSON formatada e, por fim, retorna a mensagem. Esta mensagem vai ser enviada para um nó de “write file” que foi também configurado com o caminho para o ficheiro que queremos escrever, escolhendo a opção de sobrescrever o conteúdo.

Esta função mencionada é a demonstrada na figura 5.4.

```

1  const locations = global.get("locations") || [];
2
3  global.set('locations', locations);
4
5  msg.payload = JSON.stringify(locations, null, 2);
6  return msg;

```

Figura 5.4 - Função para enviar os dados para escrita

O resultado deste *flow* irá ser demonstrado na próxima secção, assim como a demonstração funcional.

5.2. Validação funcional

Para realizarmos a validação funcional da aplicação, realizámos uma medição no espaço exterior do ISEL e, num outro momento, na zona de residência de um dos elementos do grupo.

Os resultados da medição no ISEL foram guardados num ficheiro JSON através do fluxo de escrever num ficheiro, tal como mencionado anteriormente, de maneira a testar se a funcionalidade de ler do ficheiro e atualizar o *dashboard* com resultados de medições de sessões anteriores funciona corretamente.

Antes de passarmos à validação funcional do *dashboard*, vamos observar o fluxo dos dados desde o dispositivo até chegar ao Node-RED, de maneira a validar se os dados estão a chegar corretamente ao Node-RED, ou seja, se estão a chegar à TTN, se estão a ser desempacotados corretamente e posteriormente se estão a chegar corretamente ao Node-RED através de MQTT.

Os dados recolhidos pelo sensor de qualidade de ar e pelo sensor GPS na zona de residência foram os ilustrados na tabela 5.1.

Tabela 5.1 - Dados recolhidos pelo dispositivo

Latitude	Longitude	eCO2	TVOC
38.8384567	-9.1003590	579	120

Estes valores foram então empacotados e enviados para a TTN via LoRaWAN, onde serão desempacotados, tal como mencionado anteriormente. Na figura 5.5 podemos ver a *uplink_message* que chegou à TTN.

The screenshot shows a table of device data for the entity 'vtp'. The columns are TIME, ENTITY ID, TYPE, DATA PREVIEW, and EVENT DETAILS. The last row shows an uplink message received at 15:41:09. The DATA PREVIEW shows a JSON object with fields like 'lat': 38.8384, 'long': -9.1003, and 'tvoc': 120. The EVENT DETAILS column shows the raw uplink message and its decoded payload, which includes the same sensor data and a timestamp of 2023-09-19T15:41:09Z.

TIME	ENTITY ID	TYPE	DATA PREVIEW	EVENT DETAILS
↑ 15:41:09	vtp-device	Forward location solved message	Latitude: 38.8384 Long: -9.1003 TVOC: 120	30 "uplink_message": { 31 "f_port": 2, 32 "f_cnt": 30, 33 "frm_payload": "Cj0owBoTxQ==", 34 "decoded_payload": { 35 "eco2": 500, 36 "lat": 38.8384, 37 "long": -9.1003, 38 "tvoc": 100 39 },

Figura 5.5 – Uplink message que chegou à TTN

Como podemos ver no *decoded_payload* da *uplink_message*, os valores que chegaram à TTN foram corretamente desempacotados e correspondem exatamente aos valores recolhidos pelos sensores, com a diferença de que, como referido anteriormente, estamos a ter uma precisão na ordem das centenas nos dados de qualidade de ar (579 corresponde ao 500 e 120 corresponde ao 100) e uma precisão de 4 casas decimais nos dados geográficos.

Passando agora para o Node-RED, vamos primeiramente verificar se os dados chegaram corretamente, verificando o nó de *debug* ligado ao MQTT in. Este resultado foi o apresentado na figura 5.6.

```

    ▶ uplink_message: object
      f_port: 2
      f_cnt: 30
      frm_payload: "CjOowBoTxQ=="
    ▶ decoded_payload: object
      eco2: 500
      lat: 38.8384
      long: -9.1003
      tvoc: 100
    ▶ rx_metadata: array[1]
  
```

Figura 5.6 - Payload do uplink_message recebido pelo Node-RED

Como podemos ver, os dados foram corretamente enviados pela TTN e recebidos pelo Node-RED.

Passando agora para a validação do *dashboard*, primeiramente iremos confirmar o conteúdo do ficheiro JSON (figura 5.7) que irá ser lido e os dados carregados para o *dashboard* assim que o Node-RED iniciar, que foi o resultado da medição no ISEL e que foi guardado pelo fluxo do Node-RED.

```

{} file.json X
C: > node-red > {} file.json > ...
1  [
2   {
3     "id": 1748013108519,
4     "device": "Sensor TTGO",
5     "lat": "38.756700",
6     "lon": "-9.116600",
7     "eco2": 600,
8     "tvoc": 100,
9     "quality": "Good",
10    "timestamp": "16:11:48"
11  }
12 ]
  
```

Figura 5.7 - Conteúdo do ficheiro com o resultado da medição no ISEL na sessão anterior

Posto isto, o que deverá aparecer quando iniciamos o Node-RED é apenas uma medição que corresponde à do ISEL e posteriormente, aparecerá mais dados à medida que forem realizadas mais medições. Nas figuras 5.8 e 5.9 é apresentado o *dashboard* quando o Node-RED foi iniciado, ou seja, antes da nova medição na zona de residência, e no momento após a nova medição chegar ao *dashboard*, respetivamente.

Nota: Na figura 5.8 clicou-se no ponto do mapa para demonstrar o pop-up que aparece com as informações. Não clicando, o resultado será o da figura 5.9.

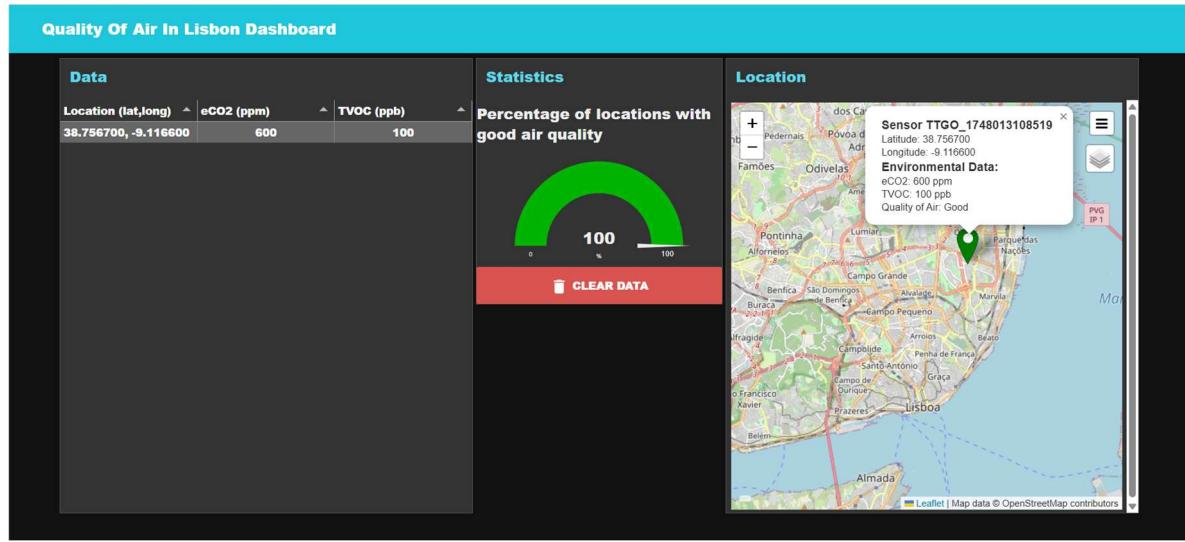


Figura 5.8 - Dashboard quando iniciamos o Node-RED

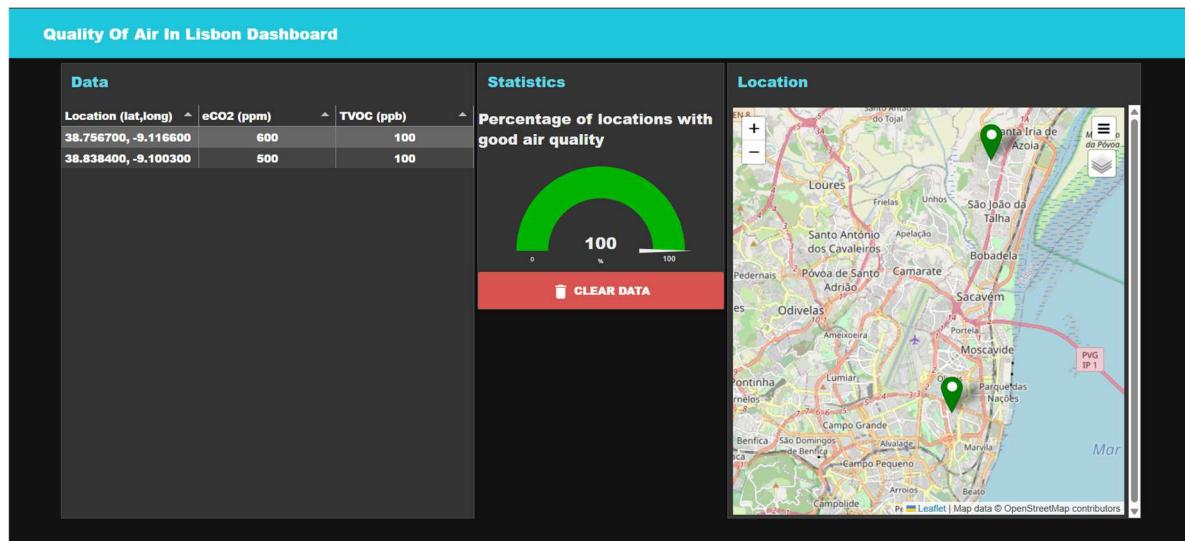


Figura 5.9 - Dashboard depois da nova medição

Após esta demonstração podemos confirmar o sucesso das funcionalidades de colocar medições anteriores no *dashboard*, com o objetivo de não pertermos medições efetuadas anteriormente, e a funcionalidade de realizar novas medições em tempo real e o *dashboard* ser atualizado ao mesmo tempo. Com isto, é possível estarmos em movimento a recolher valores de

um determinado ponto geográfico e, a cada 5 minutos, é enviado para o *dashboard* novos dados de qualidade de ar desse mesmo ponto, atualizando a tabela de valores, o gráfico e o mapa.

Por fim, resta-nos apenas demonstrar a funcionalidade de escrever no ficheiro JSON os pontos que estão no mapa, guardando assim esses valores para sessões de medição futuras. Após clicarmos no nó de *inject* denominado “Save” que inicia o fluxo de escrever no ficheiro, abrimos o ficheiro e o conteúdo é o demonstrado na figura 5.10.

```

file.json  x
C: > node-red > file.json > ...
1  [
2    {
3      "id": 1748013108519,
4      "device": "Sensor TTGO",
5      "lat": "38.756700",
6      "lon": "-9.116600",
7      "eco2": 600,
8      "tvoc": 100,
9      "quality": "Good",
10     "timestamp": "16:11:48"
11   },
12   {
13     "id": 1748100256676,
14     "device": "Sensor TTGO",
15     "lat": "38.838400",
16     "lon": "-9.100300",
17     "eco2": 500,
18     "tvoc": 100,
19     "quality": "Good",
20     "timestamp": "16:24:16"
21   }
22 ]

```

Figura 5.10 - Ficheiro após escrita na nova sessão

Nota: Vale ressaltar que estas medições foram realizadas em dias distintos, daí a proximidade do horário de medição.

Por fim, adicionámos também um fluxo que cria um botão de “*clear data*” para se quisermos limpar os valores do *dashboard*, ficando o resultado como ilustrado na figura 5.11.

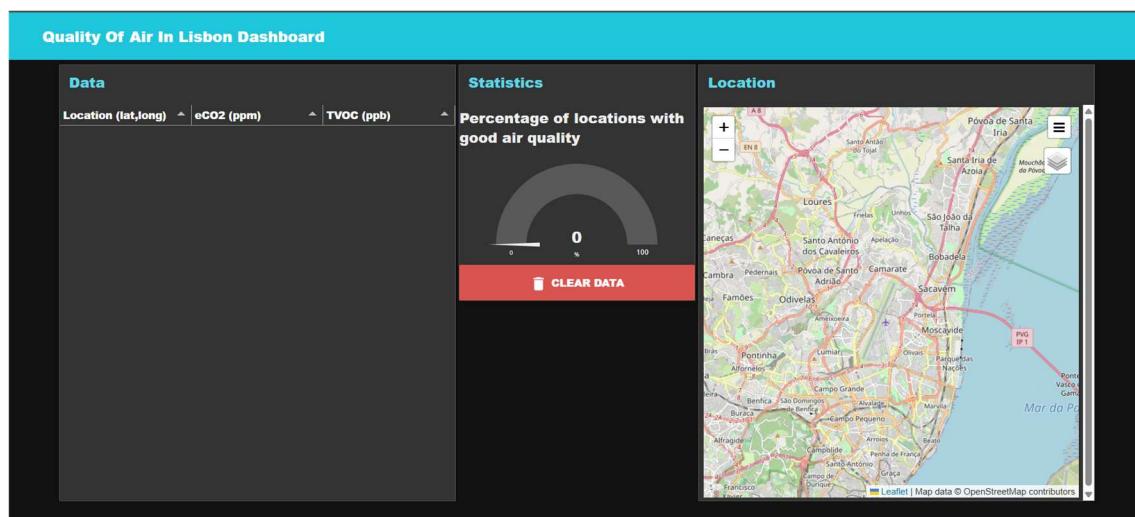


Figura 5.11 - Resultado do clear data

6. Interface da Aplicação e Validação Funcional com o Docker

6.1. Data Model e Metadata

Antes de passarmos para a interface da aplicação e validação funcional é necessário estudar e escolher um modelo de dados para representar os dados que irão ser utilizados.

Assim, de maneira a representar as medições de qualidade de ar no *Orion Context Broker*, foi adotado um modelo de dados alinhado com os *FIWARE data models*, embora simplificado para este projeto.

O *data model* utilizado como base foi obtido a partir da plataforma smartdatamodels.org, dentro do domínio “*Smart Environment*”, que pode ser acedido aqui.

Neste domínio, foi encontrando o modelo “*Environment*”, que era o que mais se adequava ao nosso projeto. Dentro deste existiam ainda vários modelos para diferentes aplicações relacionadas com o ambiente, mas o que se enquadrava no nosso projeto era o “*AirQualityObserved*”. Este modelo fornece vários exemplos para representação das medições, entre os quais foi escolhido o formato normalizado para NGSI-v2, por ser mais compacto e incluir todos os campos necessários. A estrutura adotada por este formato segue o padrão de pares “*type-value*”, onde *type* especifica o tipo de dado (por exemplo, *Number*) e *value* representa o respetivo valor.

Para esta aplicação, o modelo foi simplificado, eliminando os campos não essenciais. Assim sendo, cada medição é representada por uma entidade do tipo “info”, com os seguintes atributos:

- **eco2**: concentração de dióxido de carbono equivalente (eCO₂), com tipo “*Number*”.
- **tvoc**: concentração total de compostos orgânicos voláteis (TVOC), com tipo “*Number*”.
- **lat**: latitude da localização geográfica, com tipo “*Number*”.
- **lon**: longitude da localização geográfica, com tipo “*Number*”.

A figura 6.1 apresenta um exemplo de entidade armazenada no *Orion Context Broker*, seguindo este modelo de dados.

```
{
  "id": "12345",
  "type": "info",
  "eco2": {
    "type": "Number",
    "value": "600",
    "metadata": {}
  },
  "lat": {
    "type": "Number",
    "value": "38.1234"
  },
  "lon": {
    "type": "Number",
    "value": "-9.1111"
  },
  "tvoc": {
    "type": "Number",
    "value": 100,
    "metadata": {}
  }
}
```

Figura 6.1 - Modelo de dados adotado

Adicionalmente, existe ainda a possibilidade de inclusão de metadados, como a unidade de medida dos valores armazenados. No entanto, optou-se por não utilizar, uma vez que as unidades são consistentes e o Node-RED encarrega-se de colocar a unidade para visualização.

Ainda assim, caso se pretenda adicionar esta informação no futuro, a estrutura NGSI-v2 permite fazê-lo facilmente, conforme o exemplo abaixo:

```
"metadata": {
  "unitCode": { "value": "ppm" }
}
```

Concluindo, este modelo é compatível com a estrutura esperada por aplicações FIWARE e pode ser facilmente alterado para incorporar outros atributos ou metadados no futuro.

Passando para a modelação da base de dados, para persistência dos dados irá ser utilizado o mongoDB, conectado ao Orion em que cada entidade é armazenada como um ficheiro json com a estrutura ilustrada anteriormente.

6.2. Interface da Aplicação

Para que fosse possível adaptar o nosso Node-RED para funcionar com o Orion e o MongoDB foi necessário fazer algumas alterações ao *flow* do capítulo anterior. Após estas alterações, o *flow* final pode ser observado na figura 6.2.

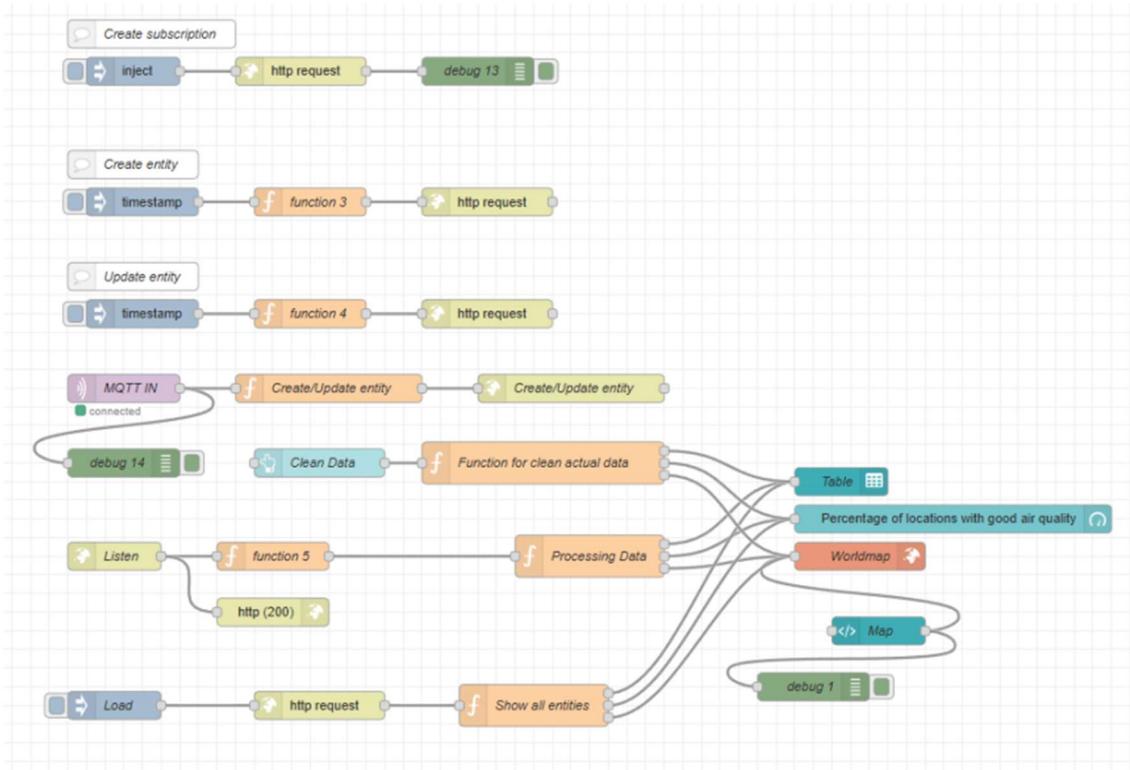


Figura 6.2 - Flow criado no Node-RED

Com o objetivo de simplificar a explicação, podemos dividir este *flow* em 4 subconjuntos, sendo estes:

- O primeiro subconjunto, que começa no nó MQTT IN e termina no nó de HTTP *Request* denominado “*Create/Update Entity*”
- O segundo subconjunto que engloba o conjunto de nós que estão rotulados como “*Create subscription*”, e que tem como objetivo a criação de uma subscrição para ser recebida uma notificação quando os dados forem alterados.
- O terceiro subconjunto que começa no nó HTTP in “*listen*” e termina nos componentes que vão constituir o *dashboard*, sendo estes a *table*, o *gauge* e o *map*, e que tem o objetivo criar o URL para receber as notificações, recolher e processar os dados adicionados ou atualizados do *Orion Context Broker* e ainda apresentar os mesmos num dashboard. Existe também um botão *clear data* que, quando pressionado, elimina os dados do dashboard.

- O último subconjunto que começa no nó de *inject* “*load*” e termina nos componentes do dashboard, tendo como objetivo carregar de todos os dados anteriormente recebidos e apresentados no *dashboard* que ficam guardados quando o Node-RED é encerrado, garantindo persistência dos dados. Assim, quando este é aberto, precisamos de carregar todas as entidades, utilizando o *load*, para as apresentar no *dashboard*.

Começando pelo primeiro conjunto de nós, o primeiro nó é um MQTT in que, como vimos anteriormente irá receber os dados provenientes da TTN. Após este nó foi colocado uma função denominada “*create/update entity*” (apresentada na figura 6.3), onde os dados são processados e organizados de forma a serem enviados para o *Orion Context Broker*, seguindo o modelo de dados apresentado na anteriormente na secção 6.1 deste documento.

```

8  const dados = msg.payload.uplink_message.decoded_payload;
9  const lat = dados.lat;
10 const lon = dados.long;
11 const eco2 = dados.eco2;
12 const tvoc = dados.tvoc;
13
14 let randomInt = Math.floor(Math.random() * (100000 - 0 + 1))
15 let randomIntStr = randomInt.toString();
16
17 const json = {
18   id: randomIntStr,
19   type: "info",
20   lat: {type: "Number", value: lat},
21   lon: {type: "Number", value: lon},
22   eco2: {type: "Number", value: eco2},
23   tvoc: {type: "Number", value: tvoc}
24 };
25
26 msg.payload = json
27
28 return msg;

```

Figura 6.3 - Código para enviar os dados com o data model escolhido para criação das entidades

Após estes serem processados, é feito um HTTP *Request*, também denominado “*create/update entity*” com o método POST para o URL “<http://orion:1026/v2/entities/>” , com a finalidade de criar uma entidade com um determinado ID e os respetivos atributos, latitude, longitude, eCO2 e TVOC.

Passando agora para o segundo subconjunto, a subscrição foi criada através de um nó de *inject* e de um de HTTP *Request*, para receber notificações assim que qualquer atributo de qualquer entidade seja modificado. O código colocado do *inject* que será enviado para o HTTP *Request* foi o demonstrado na figura 6.4:

```

1  {
2      "description": "Get lat, lon, eco2, tvoc that were updated from all entities",
3      "subject": {
4          "entities": [
5              {
6                  "idPattern": ".*"
7              }
8          ],
9          "condition": {
10             "attrs": ["lat","lon","eco2","tvoc"]
11         }
12     },
13     "notification": {
14         "http": {
15             "url": "http://node-red:1880/notification"
16         },
17         "attrs": ["lat","lon","eco2","tvoc"]
18     }
19 }

```

Figura 6.4 - Função para criação da subscrição

Desta forma, conseguimos atingir o nosso objetivo através da opção *idPattern* com ‘.*’, onde configuramos o URL “<http://node-red:1880/notification>” para receber as notificações. Este pedido de subscrição é feito com um POST para o URL “<http://orion:1026/v2/subscriptions>”, que é o URL utilizado para gerir as subscrições do Orion.

Nota: Para aceder ao *Context Broker*, pode ser utilizado o URL “<http://localhost:10026/v2/entities>” para aceder a todas as entidades e o URL “<http://localhost:10026/v2/subscriptions>” para aceder às subscrições ativas. O dashboard pode ser acedido através do URL “<http://localhost:1880/ui>”.

Passando para o terceiro subconjunto, agora que criámos a subscrição, temos de criar o URL que configurámos na subscrição, de maneira que as notificações cheguem ao Node-RED. Para isso utilizámos um HTTP In denominado “*Listen*” com o método POST como apresentado na figura 6.5. Já estando no Node-RED foi apenas necessário colocar no URL o “/notification”.

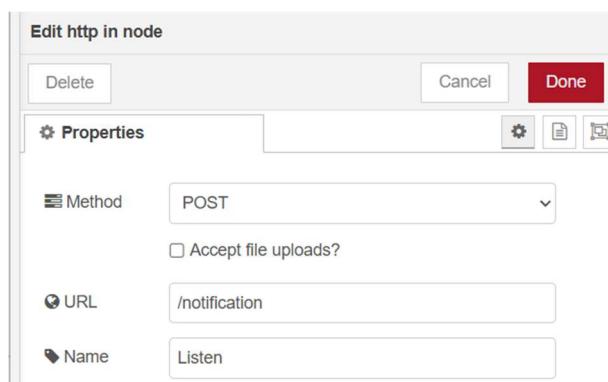


Figura 6.5 - HTTP in "Listen" para criar o URL e o Node-RED escutar as notificações

Com a notificação criada, quando os dados em qualquer entidade são alterados e recebidos através de uma notificação pelo HTTP in “*listen*”, estes passam por uma função denominada “*processing data*” que extrai os seus atributos e, de seguida, faz a formatação dos dados criando três outputs diferentes: um para apresentar a localização (latitude e longitude) com os respetivos eCO2 e TVOC numa tabela, outro para apresentar a percentagem de localizações com uma boa qualidade de ar em comparação com as com uma má qualidade do ar e, por fim, um para ser possível apresentar as localizações e respetivos valores recolhidos num mapa global.

O último subconjunto, que é o que começa no nó de *inject* denominado “*load*” foi implementado com um HTTP *Request* com o método GET para o URL “<http://orion:1026/v2/entities/>”, para que consigamos recolher todas as entidades que já estão presentes no *Orion Context Broker*, ou seja, medições que já tinham sido realizadas. Após a recolha, foi implementada uma função para processar os dados e enviá-los para o dashboard. A função implementada apresenta-se na figura 6.6.

```

3  let dados = msg.payload || [];
4  global.set('locations', []);
5
6  if (Array.isArray(dados)) {
7    let locations = global.get('locations') || [];
8    let tableData = [];
9    let mapPoints = [];
10   let goodQualityCount = 0;
11
12   for (let i = 0; i < dados.length; i++) {
13     const lat = dados[i].lat.value;
14     const lon = dados[i].lon.value;
15     const eco2 = dados[i].eco2.value;
16     const tvoc = dados[i].tvoc.value;
17
18     const goodQuality = eco2 <= 800 && tvoc <= 200;
19     const quality = goodQuality ? "Good" : "Bad";
20
21     const newPoint = {
22       id: Date.now() + i,
23       device: 'Sensor TTGO',
24       lat: lat.toFixed(4),
25       lon: lon.toFixed(4),
26       eco2: eco2,
27       tvoc: tvoc,
28       quality: quality,
29       timestamp: new Date().toLocaleTimeString()
30     };
31
32   }
33
34   locations.push(newPoint);
35
36   tableData.push({
37     "localização (lat, long)": `${newPoint.lat}, ${newPoint.lon}`,
38     "eCO2 (ppm)": newPoint.eco2,
39     "TVOC (ppb)": newPoint.tvoc,
40   });
41
42   mapPoints.push({
43     lat: parseFloat(newPoint.lat),
44     lon: parseFloat(newPoint.lon),
45     name: `${newPoint.device}_${newPoint.id}`,
46     icon: "circle",
47     iconColor: newPoint.quality === "Good" ? "green" : "red",
48     popup: `Latitude: ${newPoint.lat} <br>Longitude: ${newPoint.lon}` +
49           `  
<b>Environmental Data:</b><br>eCO2: ${newPoint.eco2} ppm<br>` +
50           `TVOC: ${newPoint.tvoc} ppb<br>Quality of Air: ${newPoint.quality}`,
51     layer: "locations"
52   });
53
54   if (goodQuality) goodQualityCount++;
55
56   global.set('locations', locations);
57
58   const qualityPercentage = Math.round((goodQualityCount / locations.length) * 100);
59
60   return [{payload: tableData}, {payload: qualityPercentage}, {payload: mapPoints}];
61 }
```

Figura 6.6 - Função implementada para recolher e representar os dados que já estavam no Orion Context Broker de sessões anteriores

6.3. Validação funcional

Para validar o funcionamento da aplicação após a integração com Docker, foi utilizado o mesmo cenário da validação funcional apresentada no Capítulo 5, composto por duas medições: uma realizada no espaço exterior do ISEL e outra, noutro momento, na zona de residência de um dos elementos do grupo.

As etapas de envio de dados e receção por parte da TTN, do envio por MQQT e da receção pelo Node-RED mantêm o comportamento demonstrado na secção 5.2 deste documento, nomeadamente nas figuras 5.5 e 5.6, pelo que não será novamente demonstrada.

Os dados recolhidos pelo sensor de qualidade de ar e pelo sensor GPS na zona de residência foram semelhantes aos que tinham sido recolhidos e demonstrados anteriormente, sendo os ilustrados na tabela 6.1.

Tabela 6.1 - Dados recolhidos pelo dispositivo

Latitude	Longitude	eCO2	TVOC
38.8384470	-9.1003610	549	150

Ao contrário do que acontecia anteriormente, os dados são agora automaticamente armazenados como entidades no *Orion Context Broker*. Como o Orion está ligado a uma base de dados MongoDB, garante-se a persistência entre sessões sem necessidade de escrita manual em ficheiro.

Passando agora para a validação funcional, começámos por confirmar o conteúdo das entidades armazenadas no *Orion Context Broker* (figura 6.7) que irá ser recolhido e carregado para o *dashboard*, assim que clicarmos no botão de load. Neste caso estará apenas a medição que foi realizada no ISEL e que, na implementação anterior, tinha sido guardada num ficheiro.



```
[{"id": "15630", "type": "info", "eco2": {"type": "Number", "value": 600, "metadata": {}}, "lat": {"type": "Number", "value": 38.7567, "metadata": {}}, "lon": {"type": "Number", "value": -9.1166, "metadata": {}}, "tvoc": {"type": "Number", "value": 100, "metadata": {}}}]
```

Figura 6.7 - Entidades armazenadas no Orion antes da nova medição

Posto isto, o que deverá aparecer quando clicarmos no botão de load é apenas uma medição que corresponde à do ISEL. Posteriormente, aparecerá mais dados à medida que forem realizadas mais medições. Nas figuras 6.8 e 6.9 é apresentado o *dashboard* antes da nova medição e no momento após a nova medição chegar ao *dashboard*, respetivamente.

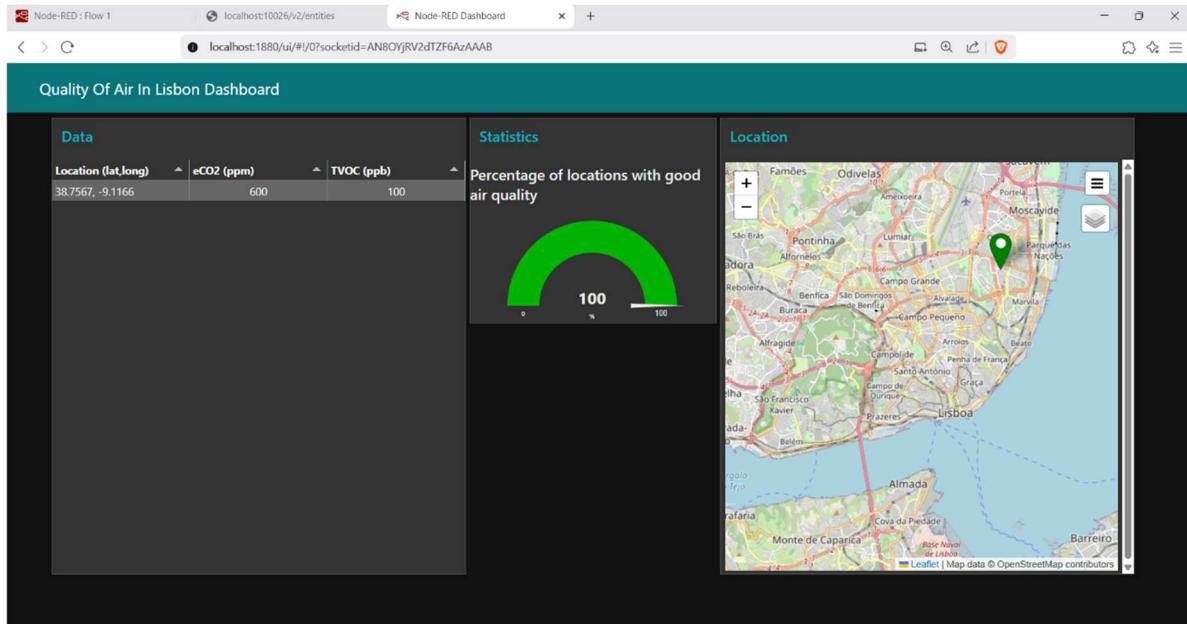


Figura 6.8 - Dashboard quando clicamos no botão de load

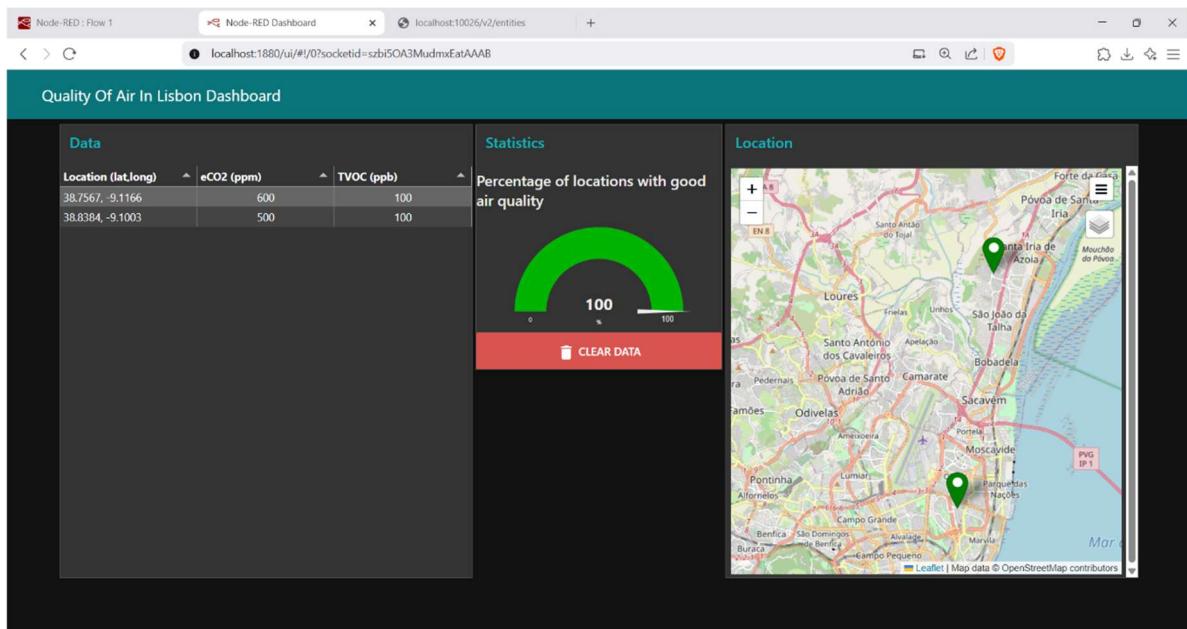


Figura 6.9 - Dashboard depois da nova medição

Com esta demonstração, é possível confirmar que:

- Medições anteriores são corretamente recuperadas numa nova sessão.
- Novas medições são apresentadas em tempo real no dashboard, com atualização do mapa, da tabela e do gráfico.

Com isto, é possível estarmos em movimento a recolher valores de um determinado ponto geográfico e, a cada 5 minutos, é enviado para o *dashboard* novos dados de qualidade de ar desse mesmo ponto, atualizando a tabela de valores, o gráfico e o mapa.

Ao contrário do que foi mostrado na validação funcional antes do Docker, neste caso já não vai ser necessário escrever num ficheiro todas as medições efetuadas, uma vez que, quando é realizada uma nova medição, essa mesma medição é automaticamente enviada para o Orion, sendo armazenada como uma nova entidade na base de dados MongoDB, garantindo assim persistência de dados entre sessões.

Na figura 6.10 é demonstrado o conteúdo das entidades após a nova medição efetuada que, como é esperado, terá a antiga medição e a nova.

```
[{"id": "15630", "type": "info", "eco2": {"type": "Number", "value": 600, "metadata": {}}, "lat": {"type": "Number", "value": 38.7567, "metadata": {}}, "lon": {"type": "Number", "value": -9.1166, "metadata": {}}, "tvoc": {"type": "Number", "value": 100, "metadata": {}}, {"id": "76642", "type": "info", "eco2": {"type": "Number", "value": 500, "metadata": {}}, "lat": {"type": "Number", "value": 38.8384, "metadata": {}}, "lon": {"type": "Number", "value": -9.1003, "metadata": {}}, "tvoc": {"type": "Number", "value": 100, "metadata": {}}}
```

Figura 6.10 – Conteúdo das entidades depois da nova medição

Por fim, verificámos a funcionalidade do botão de “*clear data*” que limpa os valores do *dashboard*, ficando o resultado como ilustrado na figura 6.11.

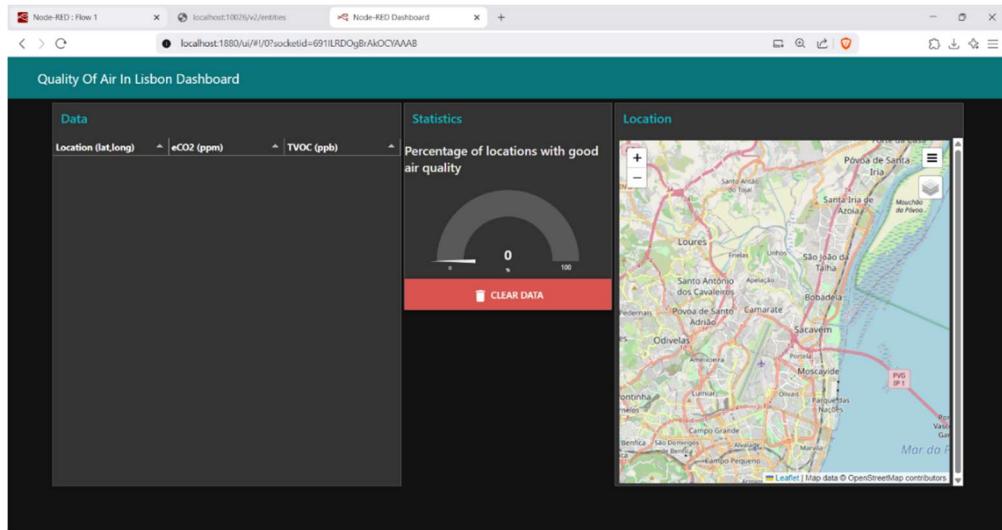


Figura 6.11 - Resultado do clear data

Posto isto, comparando com a implementação anterior, sem a utilização do Docker, com adoção de uma arquitetura baseada em containers (Docker) e a utilização do *Orion Context Broker* com persistência em MongoDB, a aplicação passou a garantir persistência entre sessões sem dependência de ficheiros locais, o que é uma melhoria significativa.

7. Manual de Utilização da Aplicação

Este capítulo apresenta um guia para a utilização do sistema de monitorização de qualidade de ar descrito neste documento, apresentando os requisitos e o funcionamento geral.

7.1. Requisitos da aplicação

Começando pelos requisitos, para que a aplicação funcione corretamente, são necessários os seguintes componentes e configurações:

HARDWARE

- Microcontrolador TTGO T-Beam com módulo GPS e LoRa integrados;
- Sensor de qualidade de ar CSS811 ligado via I2C ao TTGO (conforme ilustrado na figura 7.1);
- Bateria para alimentação portátil (opcional, mas recomendado para maior portabilidade);
- Gateway LoRaWAN registada na TTN.

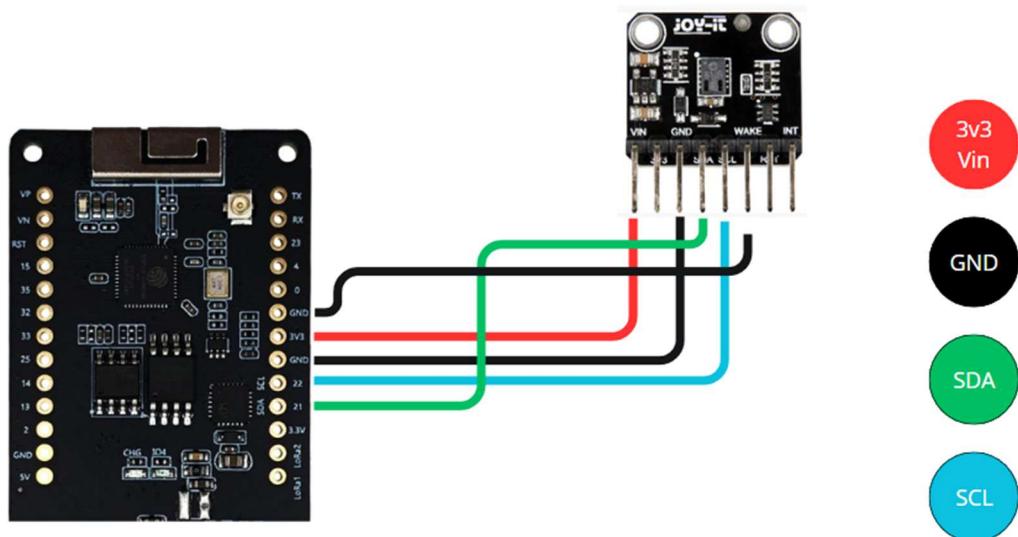


Figura 7.1 - Ligações para comunicação I2C com o sensor

SOFTWARE

- Código em MicroPython carregado para o TTGO;
- Conta na TTN com o dispositivo configurado de acordo com os dados da TTN;
- Docker desktop com o orion, mongoDB e o node-red, com o fluxo demonstrado na secção anterior, para visualização dos dados no *dashboard*.

7.2. Funcionamento

Para o funcionamento geral da aplicação, o sistema descrito realiza as seguintes etapas:

1. Com o código carregado no TTGO e este alimentado, quando o botão de reset (RST) do dispositivo for pressionado, este corre o código, ativando os módulos GPS, LoRa e o sensor CCS811.
2. Após a inicialização dos componentes, durante vinte minutos o dispositivo apenas lê dados, sem os processar, para dar o devido tempo para todos os módulos estabilizarem e apresentarem valores válidos.
3. Após este tempo, o dispositivo recolhe os dados de qualidade de ar provenientes do sensor CCS881 durante cinco minutos, armazenando os mesmos em dois arrays, um para o eCO2 e outro para o TVOC.
4. Quando passarem os cinco minutos, é realizada a média dos valores presentes nos arrays, ficando com apenas um valor de cada variável (eCO2 e TVOC). Neste momento é também recolhida a localização geográfica (latitude e longitude) do ponto onde foram recolhidos estes dados de qualidade de ar.
5. Após recolher estes 4 dados, os mesmos são empacotados em 7 byte, tal como demonstrado anteriormente, e enviados via LoRaWAN para uma *gateway* da TTN (ler nota).
6. Quando os dados chegam à TTN, estes são desempacotados e enviados por MQTT para a plataforma de visualização IoT, neste caso o Node-RED.
7. Por fim, no Node-RED os dados são processados e, após passar pelo fluxo criado e demonstrado anteriormente, aparecerão no *dashboard* em forma de tabela, gráfico e ainda um ponto no mapa.
8. Depois dos dados apresentados, repete-se tudo de novo desde o ponto 3.

Nota: Para um envio e receção corretos dos dados pela TTN é crucial que, no código carregado para o dispositivo, esteja as informações corretas presentes na TTN, como as *keys* e o *device_addr*. É também crucial e imprescindível que exista uma gateway da TTN próxima do local de envio para que os dados cheguem à TTN.

Ao aceder ao *dashboard*, através do endereço IP ou domínio configurado no Node-RED que normalmente seria “<http://localhost:1880/ui>”, podemos observar os dados em cada ponto geográfico e aferir a qualidade de ar, em que valores de eCO2 menores que 800 e de TVOC menores que 200 traduzem uma boa qualidade de ar.

Resumindo, este sistema é de uma utilização muito simples, em que o utilizador apenas necessita de conectar corretamente o sensor CCS881 ao dispositivo, alimentar o dispositivo e clicar no botão de RST do mesmo. Após isto, e após passarem vinte minutos, os dados começarão a aparecer no *dashboard* a cada cinco minutos, se a transmissão pelo dispositivo tiver ocorrido com sucesso. Estes dados podem ser visualizados e serão guardados em entidades no orion e, consequentemente, uma base de dados, para persistência dos dados.

Ter em atenção para o bom funcionamento do sistema:

- Garantir que a bateria está carregada e o dispositivo corretamente alimentado;
- Verificar se existe uma *gateway* próxima do local de medição;
- Evitar obstruções do sinal GPS;
- Verificar que o sensor CCS811 está corretamente ligado ao dispositivo.

8. Referências

- [1] *A qualidade do ar na europa continua a melhorar, Mas Os Níveis de Poluição Continuam a ser perigosos em Muitas Zonas* (2024) European Environment Agency. Disponível em: <https://www.eea.europa.eu/pt/highlights/a-qualidade-do-ar-na> (Acedido: 20 Março 2025).
- [2] Labs, I. (2024) *Monitoramento da Qualidade do ar Interno em Sala de Aula, IoT Labs - American Tower do Brasil - Ecossistema de Soluções para IoT*. Disponível em: <https://iot-labs.io/solucao-covid-19-monitoramento-da-qualidade-do-ar-interno/> (Acedido: 20 Março 2025).
- [3] What are Lora and Lorawan? (2024) The Things Network. Disponível em: <https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/> (Acedido: 22 Abril 2025).
- [4] The Things Network Lisbon, The Things Network. Disponível em: <https://www.thethingsnetwork.org/community/lisbon/> (Acedido: 30 April 2025).
- [5] Arjanvanb (2020) *Best practices to limit application payloads*, The Things Network. Disponível em: <https://www.thethingsnetwork.org/forum/t/best-practices-to-limit-application-payloads/1302> (Acedido: 22 Abril 2025).
- [6] EN 300 220-2 - v3.2.1. Disponível em: https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.02.01_60/en_300220_02v030201p.pdf (Acedido: 23 Abril 2025).
- [7] Duty cycle The Things Network. Disponível em: <https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/> (Acedido: 23 Abril 2025).
- [8] *Lora® Calculator smtc*. Disponível em: <https://www.semtech.com/design-support/lora-calculator> (Acedido: 24 Abril 2025).