Licenciatura

Engenharia de Sistemas Informáticos

Integração de Sistemas de Informação

Trabalho Prático 1

Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Gonçalo Tierri Martinho Gonçalves

23020

Barcelos, Outubro de 2025

Licenciatura

Engenharia de Sistemas Informáticos

Integração de Sistemas de Informação

Trabalho Prático 1

Instituto Politécnico do Cávado e do Ave

Escola Superior de Tecnologia

Gonçalo Tierri Martinho Gonçalves

23020

Barcelos, Outubro de 2025

Índice

Índice de Figuras

1. Introdução

Para a Unidade Curricular de Integração de Sistemas de Informação, foi pedido aos alunos para desenvolverem processos *ETL* que integrem dados de um sistema, e os transferem para outro, de modo a consolidar os conhecimentos do estudante nesta área da engenharia informática.

2. Tema do Projeto

Na atualidade, é cada vez mais necessário consciencializar a população sobre a poluição aérea, pois é algo que afeta a todos, porém pode passar despercebida no dia-a-dia. Este projeto tem como objetivo principal obter dados de sensores de gases e partículas poluidoras públicos, NO_2 (Dióxido de Nitrogénio) e *PM10* (pólen, poeira, fuligem, e outras partículas metálicas) e tratar os resultados para obter um histórico dos últimos dois anos das medições, e um índex que mostra os dados atuais, junto com um avaliador em texto, para fácil compreensão da qualidade atual do ar para um utilizador que possa não ter conhecimento o suficiente para interpretar os dados sozinho.

3. Ferramentas Utilizadas

Para viabilizar este projeto, as ferramentas utilizadas foram:

Knime – A ferramenta utilizada para criar o *workflow* que extrai os dados e os transforma, e envia para o Node-Red.

Node-Red – A ferramenta utilizada para visualizar os dados de maneira fácil.

OpenAQ – A *API* pública e gratuita que contém dados históricos e atuais de sensores de *PM10* e NO_2 em Braga e ao redor do mundo.

4. Processo ETL

4.1. Knime

O processo com os dados é iniciado no Knime e está divido em três principais etapas:

Obtenção

↓

Limpeza e tratamento

↓

Envio para o Node-Red

Em mais detalhe,

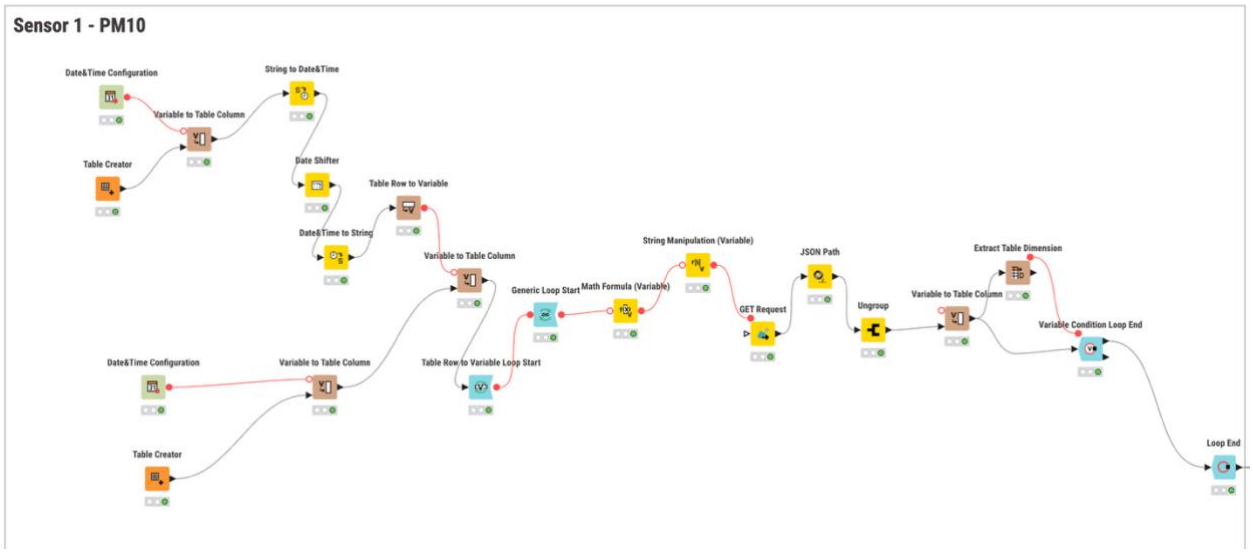


Figura 1 - Obtenção de Dados para o Sensor de PM10

O que está a acontecer nesta etapa é a criação de duas tabelas, e é adicionada a data atual, e informações sobre o sensor ao qual vamos recolher dados (o tipo de dado, e o ID do sensor) a uma delas, enquanto a outra tabela vai receber apenas a data atual.

A segunda tabela tem a data convertida para o formato DateTime, e depois são deduzidos 2 anos a essa data. Esta será a data até onde vão ser obtidos os dados.

As duas tabelas são concatenadas numa só, de modo a facilitar a leitura da *flow*

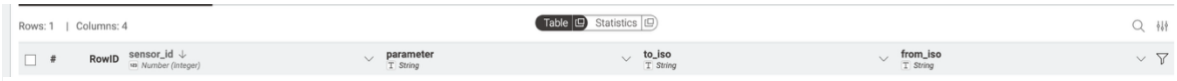
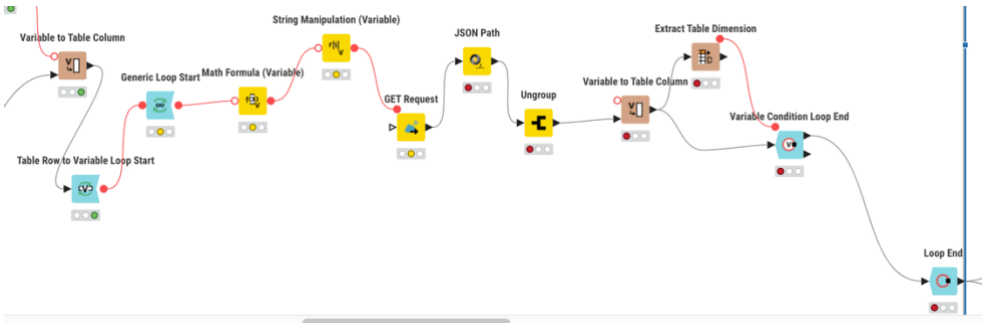


Figura 2 - Tabela concatenada

Como a API tem limite da quantidade de dados que pode transmitir por pedido (1000 medições), e funciona por páginas ou seja, a página 1 transmite os primeiros 1000 resultados, e página 2 transmite os segundos 1000 resultados,

e assim continuamente. Decidi criar um loop que faz pedidos à API utilizando as variáveis que eu introduzi nas tabelas como argumentos no URL.



Este *Table Row to Variable Loop Start* lê a tabela que contém todos os dados necessários para o pedido na *API* (data atual, data de há 2 anos atrás, *ID* do sensor e o dado medido) e torna-os em *Flow Variables*, adicionalmente também cria uma variável automaticamente, denominada "*CurrentIteration*", que é útil para o último argumento necessário no *URL* da *API*. Como o CurrentIteration começa com o valor 0, e a contagem das páginas começa a 1, é necessário usar uma fórmula matemática para obter o valor do argumento *page* no URL da api. Uma simples fórmula que soma "CurrentIteration + 1" é suficiente para tal objetivo.

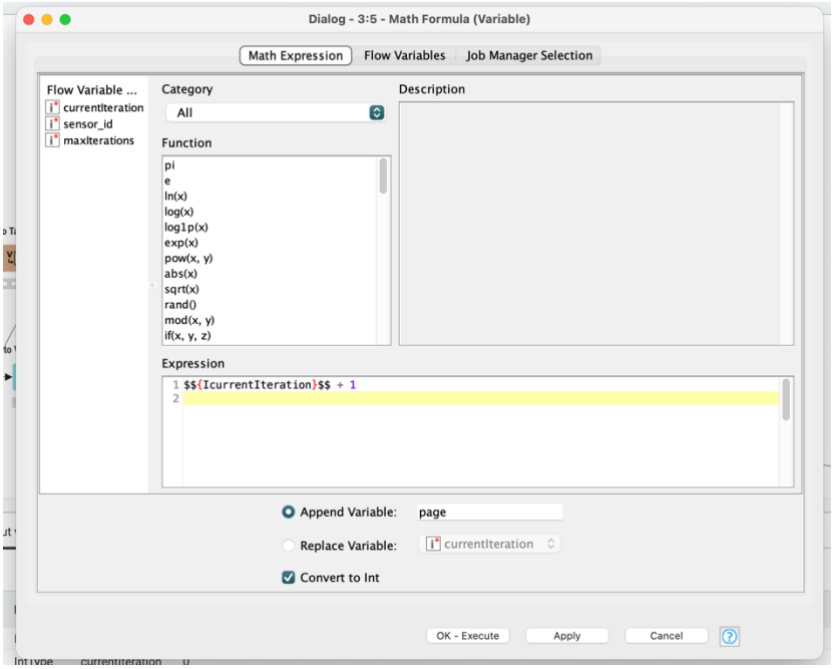


Figura 3 - Math Formula para obter o valor de Page

Após o cálculo da nova variável inteira "*page*", é necessário efetivamente criar o *URL* que vai ser usado no pedido *GET* à *API* da *OpenAQ*.

O *URL* é esperado no seguinte formato:

https://api.openaq.org/v3/sensors/**sensor_id**/measurements?limit=1000&datetime_from=**from_iso**Z&datetime_to=**to_iso**Z&page=**page**

Com o texto a negrito sendo o valor das variáveis. Para obter este *URL* é necessário juntar as variáveis no formato correto, usando uma simples *script* em *Java*. A *script* abaixo serve para tal.

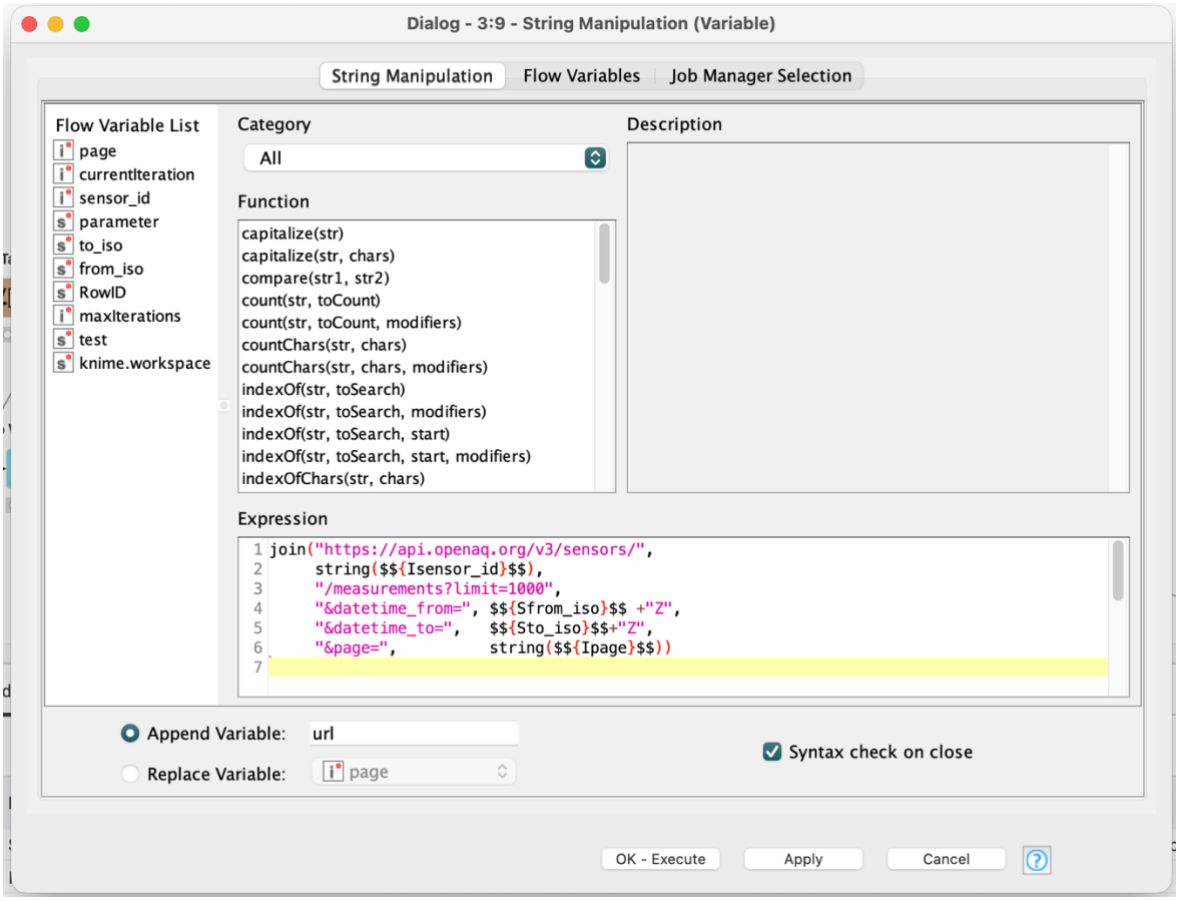


Figura 4 - Exportar o URL criado

Após a criação da *string url*, é apenas necessário fazer um pedido à *API*. Usando o nó *GET Request*, e injetando a variável *url* como o *URL* constante e enviando a chave da *API* no *Header* do pedido.

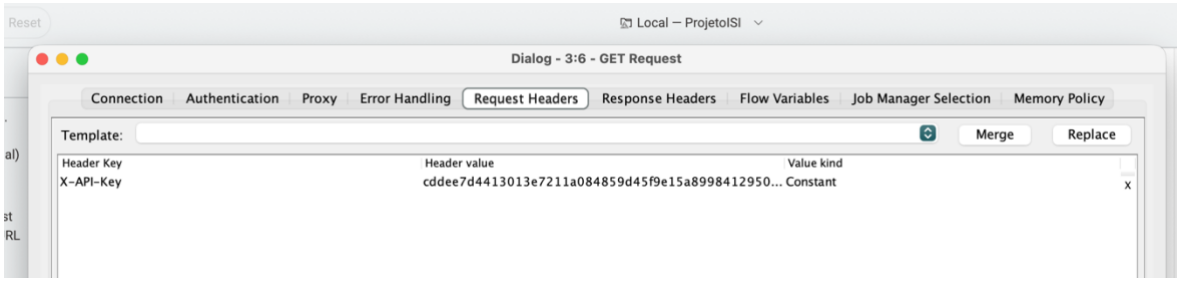


Figura 6 - Request Header

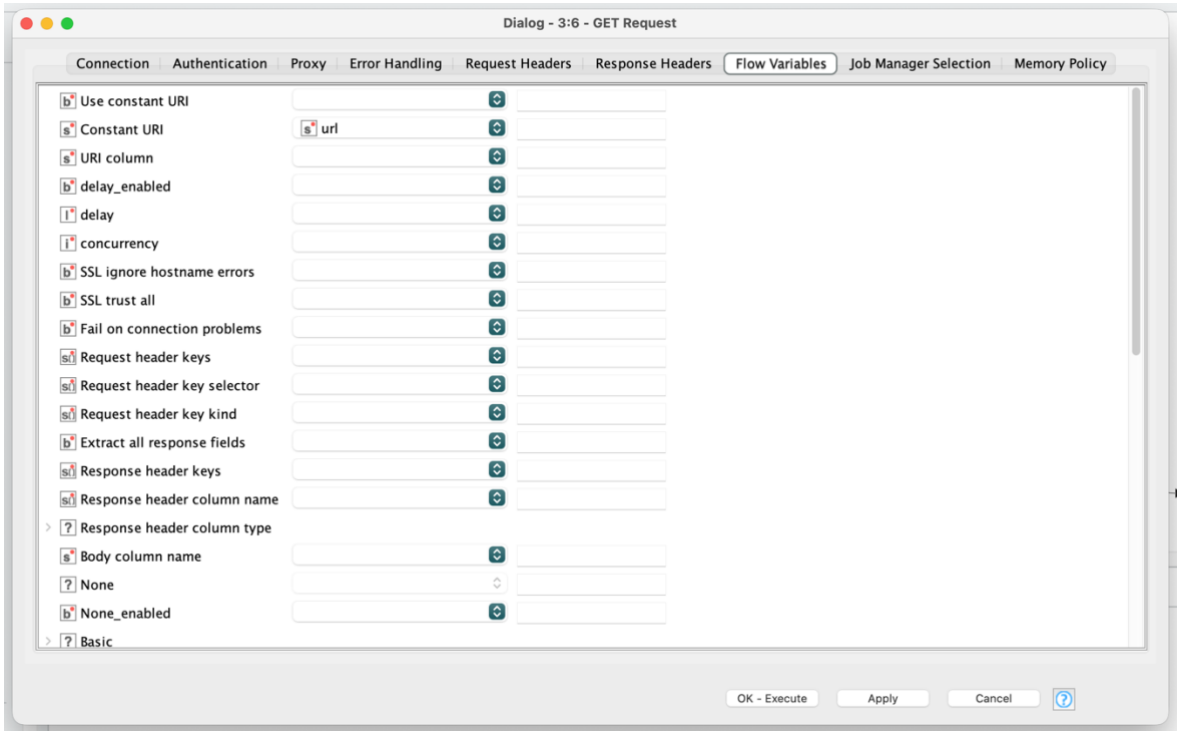


Figura 5 - Variável de Fluxo url usada como constante

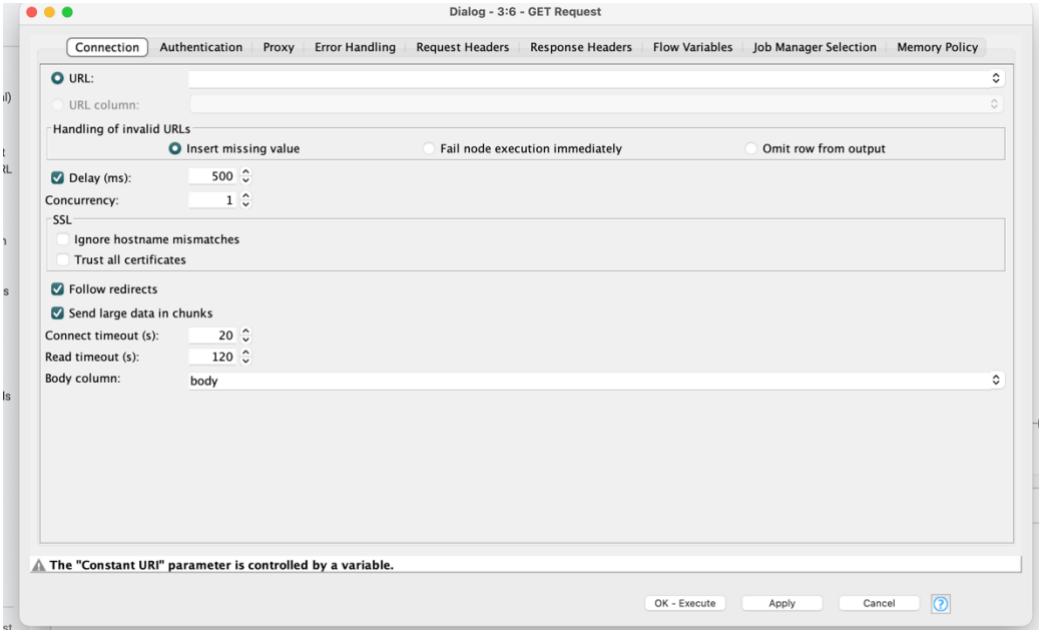


Figura 7- Configuração do pedido GET

Após receber a resposta em formato *JSON* no corpo da resposta, é necessário filtrar pelos campos mais importantes da resposta, pois há dados que não são interessantes para o escopo deste projeto. Para isso é utilizado um nó *JSON Path*, configurado para recolher os dados importantes numa tabela.

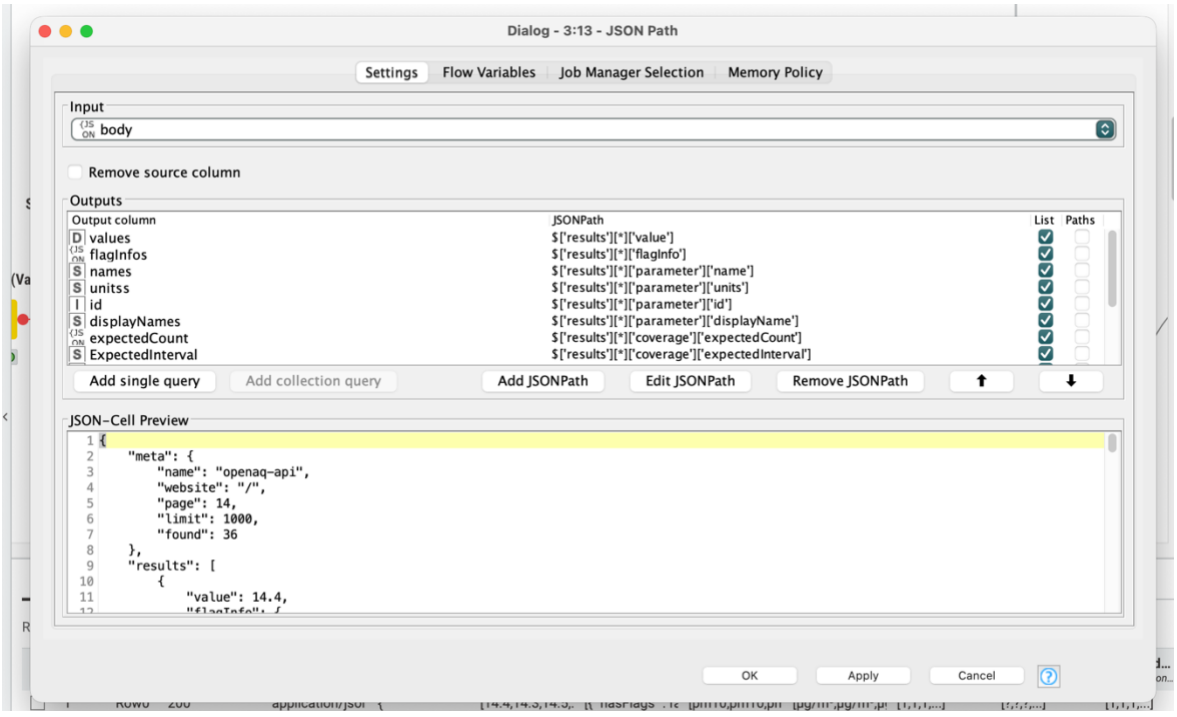


Figura 8- Configuração do nó JSON Path

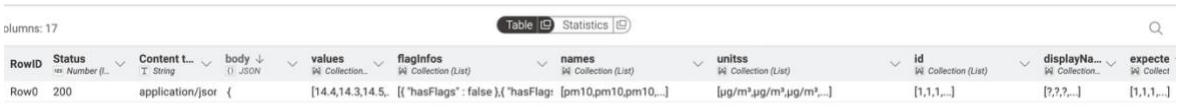


Figura 9 - Parte da tabela após a execução do nó

Ora, como se pode ver na tabela acima, só existe uma linha, que contém *arrays* com todos os valores para cada uma das leituras realizadas. Isto ainda não é útil, então terá de ser executado um nó Ungroup que separa todos os arrays nas suas linhas individuais, aproveitamos e adicionamos outro nó após esse que atribui as variáveis de fluxo *sensor_id, from_iso, to_iso* e *parameter* às suas próprias colunas na tabela resultante.

Este loop vai-se repetir, adicionando sempre as leituras recebidas à tabela que estamos a construir, até eventualmente receber menos de 1000 leituras numa das respostas. Isto significa que é a última página e que o loop deve terminar após tal. Os nós abaixo tratam disso.

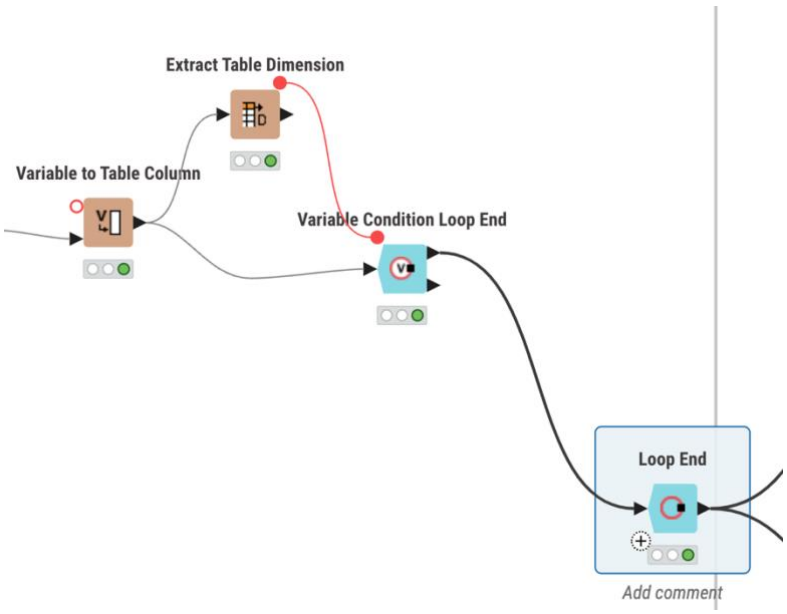


Figura 10 - Fim do Loop

É importante mencionar que tudo o que acabou de ser descrito só está a acontecer para um dos dois sensores que este projeto utiliza. Para obter os dados do outro sensor, um fluxo exatamente igual corre em paralelo a este, onde as únicas coisas que mudam são o *sensor_id* e o parameter na tabela inicial. Isto pode ser escalado e podem ser criados mais fluxos apenas copiando e colando o fluxo e colocando valores diferentes nessas tabelas para obter mais dados caso necessário.

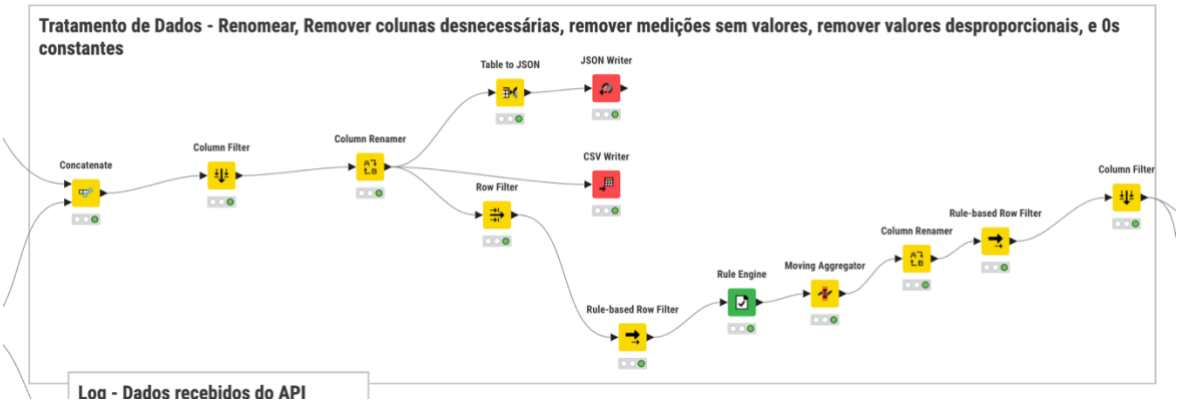


Figura 11 - Tratamento de dados

Para o tratamento dos dados recebidos, a primeira coisa a fazer é concatenar todas as tabelas recebidas, uma vez que o formato das tabelas é igual, não haverá colunas vazias dependendo dos dados recebidos.

Após esse primeiro passo, vamos remover colunas desnecessárias (Status, Body, flagInfos, id, displayNames, expectedCount, ..) que não têm dados interessantes para o projeto.

Depois, por motivos de facilitar a leitura, renomeamos colunas

Unitss -> Units

Names -> MeasuredGas

Utcs -> readStart

Utcs (#1) -> readFinish

Feito isso, usamos um filtro para remover linhas cuja leitura seja menor que 0. A *OpenAQ* marca a leitura com o valor -1 caso seja alguma leitura que não tenha sucedido por algum motivo. Depois desse filtro inicial, fazemos um filtro mais complexo que remove leituras que têm valores que são considerados impossíveis. Neste caso, *PM10* acima de 300 e NO_2 acima de 400.

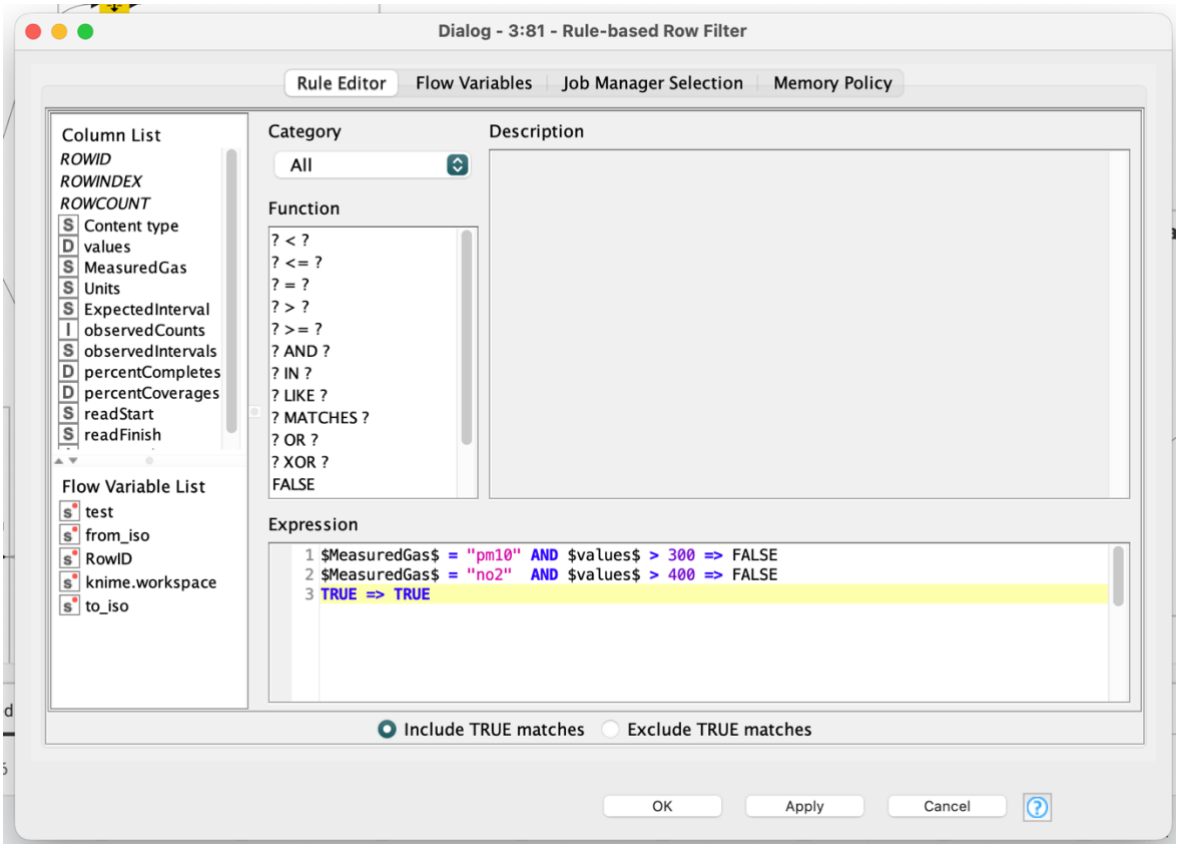


Figura 12 - Filtro que remove leituras impossíveis

Removendo esses valores abismais, é boa ideia remover leituras que tenham o valor de 0 durante demasiado tempo seguido. Tal valor é altamente improvável numa cidade e muito possivelmente um erro no sensor. Este filtro é mais complexo e requer quatro nós para ser possível

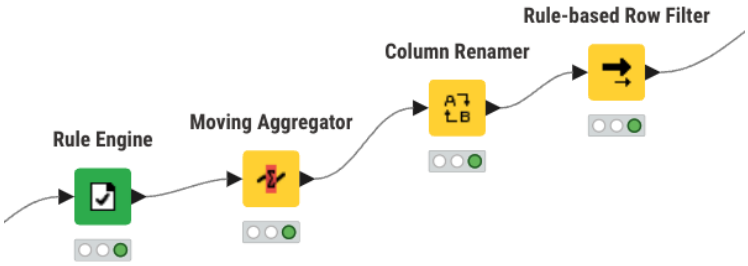


Figura 13 - Nós para filtrar 0s constantes

A *rule engine* determina se um valor é 0 ou não. Apende esse *int* (1 ou 0) numa nova coluna associada à medição.

O *moving aggregator* soma o número de vezes que um *is_zero* aparece junto e apende esse valor numa nova coluna. O *column renamer* muda o nome dessa coluna gerada pelo *moving aggregator* para *zero_run6.* Finalmente, o *rule-based row filter* remove qualquer leitura cujo valor de *zero_run6* seja maior ou igual a 6.

Após tais operações, esta tabela está pronta para ser enviada para o Node-Red, onde vai ser visualizada.

Para tal, convertemos a tabela para *JSON* e enviamos num *POST Request* para o endpoint da API do servidor Node-Red previamente configurado e a executar.

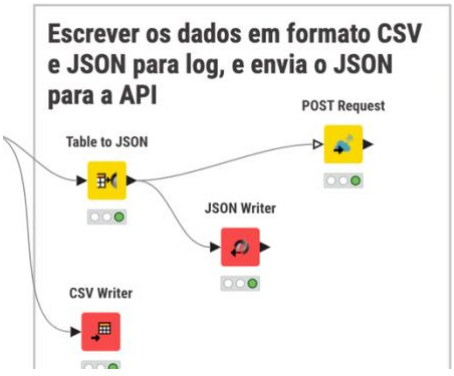


Figura 14 - Envio dos dados para o Node-Red

4.2. Node-Red

Este é o *layout* da *workflow* no Node-Red:

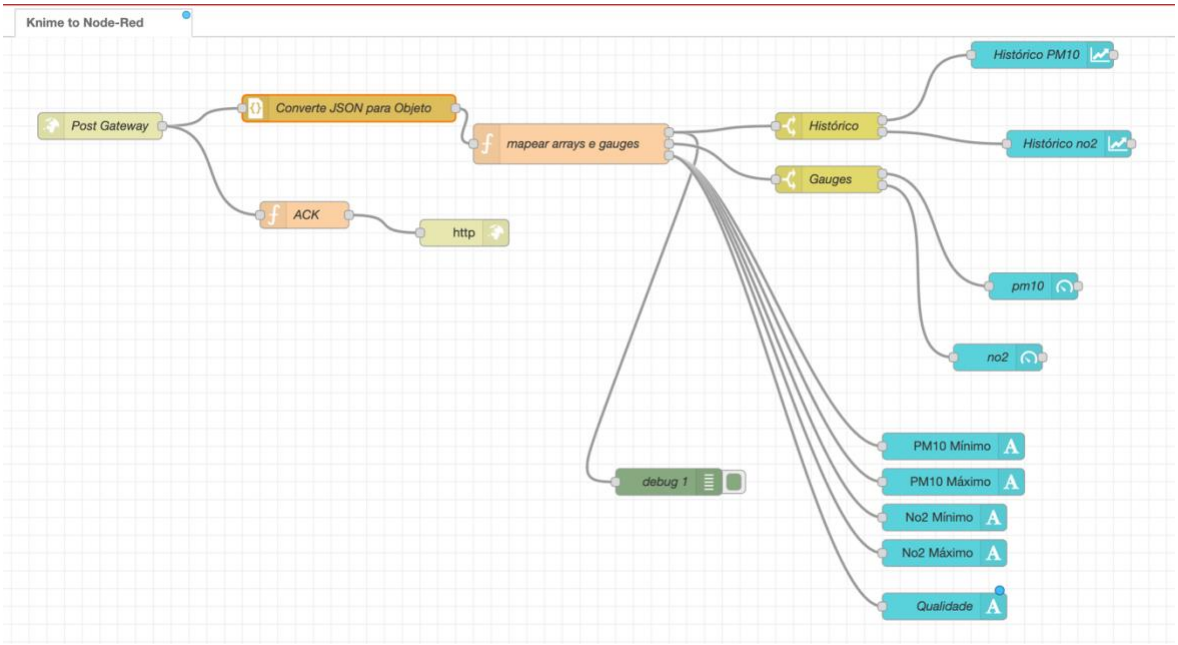


Figura 15 - Layout Node-Red

Essencialmente, este fluxo abre uma porta no *URL* "<ip>/aq" para receber o *JSON*, onde após receber uma mensagem por essa porta, converte o corpo do pedido para um objeto *Javascript*, e ao mesmo tempo envia uma resposta com o código 200, para garantir ao remetente que a mensagem foi recebida.

Após converter para objeto, é necessário fazer repartir os dados necessários para cada gráfico.

Precisamos de separar os dados mais recentes, para mostrar no medidor, e de visualizar os dados históricos num gráfico. Para além disso, temos de interpretar os dados atuais, numa maneira fácil de compreender.

Escolhi para então mapear os valores lidos na escala definida pelos padrões europeus da qualidade do ar EAQ.

POLLUTANT	INDEX LEVEL *(based on polluant concentrations in µg/m3)*					
	1 Very good	2 Good	3 Medium	4 Poor	5 Very Poor	6 Extremely Poor
Ozone (O_3)	0-50	50-100	100-130	130-240	240-380	380-800
Nitrogen dioxide (NO_2)	0-40	40-90	90-120	120-230	230-340	340-1000
Sulphur dioxide (So_2)	0-100	100-200	200-350	350-500	500-750	750-1250
Particules less than 10 µm (PM_{10})	0-20	20-40	40-50	50-100	100-150	150-1200
Particules less than 2.5 µm ($PM_{2.5}$)	0-10	10-20	20-25	25-50	50-75	75-800

Note: PM10 and PM2.5 values are based on 24-hour running means

Figura 16 - Tabela com os níveis da qualidade do ar

Para calcular o índice de qualidade do ar (EAQI) na *dashboard*, definimos para cada poluente os limiares em µg/m³ que delimitam seis classes ("Muito Boa", "Boa", "Moderada", "Má", "Muito Má", "Extremamente Má"); uma função percorre estes limites em ordem crescente e atribui ao valor observado o primeiro escalão cujo topo é superior ao próprio valor (produzindo um nível 1–6 e o respetivo rótulo, ficando no nível 6 se exceder o último limite). O valor de referência do *PM10* é a média dos registos disponíveis numa janela móvel de 24 horas, enquanto para o NO_2 é usado o último valor horário disponível. Se não houver dados na janela, o índice desse poluente fica nulo.

Para o indicador global de qualidade do ar, considera-se o pior dos dois, preservando a interpretação e comparabilidade do EAQ.

```
const EAQI = {
  PM10: [0, 20, 40, 50, 100, 150, 1200],
  NO2:  [0, 40, 90, 120, 230, 340, 1000]
};
const LABELS = ['Muito Boa','Boa','Moderada','Má','Muito Má','Extremamente Má'];

function eIndex(val, limits) {
  if (val == null || !Number.isFinite(val)) return null;
  for (let i = 0; i < limits.length - 1; i++) {
    if (val < limits[i + 1]) return { level: i + 1, label: LABELS[i] };
  }
  return { level: 6, label: LABELS[5] };
}

// PM10: média 24h
const t24 = Date.now() - 24 * 3600 * 1000;
const pm10_24h_vals = (byGas.PM10 || [])
  .filter(p => p.x >= t24 && p.y > MIN_POS)
  .map(p => p.y);
const pm10_avg24 = pm10_24h_vals.length
  ? pm10_24h_vals.reduce((a, b) => a + b, 0) / pm10_24h_vals.length
  : null;

// NO2: último valor (1h)
const no2_last = (byGas.NO2 || []).at(-1)?.y ?? null;

// construir stats
const stats = {
  windowDays: 730,
  PM10: minmax(byGas.PM10 || []) || {},
  NO2:  minmax(byGas.NO2  || []) || {},
  now:  { PM10_24h_avg: pm10_avg24, NO2_1h: no2_last },
  eaqi: {
    PM10: eIndex(pm10_avg24, EAQI.PM10),
    NO2:  eIndex(no2_last,  EAQI.NO2)
  }
};
//  índice do pior
const levels = [stats.eaqi.PM10?.level || 0, stats.eaqi.NO2?.level || 0];
const worst  = Math.max(...levels);
stats.eaqi.combined = worst ? { level: worst, label: LABELS[worst - 1] } : null;
```

Figura 17 - Código responsável pela classificação e

Também temos os medidores dos dados atuais, que recebem o valor mais atual possível, e os gráficos com os dados históricos.

O medidor usa uma janela das últimas vinte e quatro horas para cada poluente, filtra as leituras desse período e envia para o *gauge* o valor mais recente disponível; se nessa janela não houver dados (por exemplo, logo após

a meia-noite), faz *fallback* para o último valor válido da série, garantindo que o mostrador nunca fica vazio. Já o histórico prepara as séries temporais para o gráfico ao agrupar e ordenar os pontos por poluente e depois reduzir a densidade com um *downsampling* simples de passo fixo (mantendo 1 em cada 3000 pontos), o que preserva a tendência visual sem sobrecarregar a interface; cada mensagem de série segue o formato esperado pelo *chart* do Node-RED, com *topic* igual ao nome do gás e *payload* como uma lista de pares {x,y} em milissegundos e µg/m³

```
// ---------- GAUGES (último de hoje) ----------
const nowTs = Date.now();
const winMs = msg.gaugeWindowMs ?? 24*3600*1000;
const since = nowTs - winMs;

const gaugeMsgs = [];
for (const [gas, pts] of Object.entries(byGas)) {
  const w = pts.filter(p => p.x >= since && Number.isFinite(p.y));
  let val = w.at(-1)?.y ?? pts.at(-1)?.y ?? null; // usa último disponível caso as ultimas 24h estejam vazias
  if (Number.isFinite(val)) gaugeMsgs.push({ topic: gas, payload: val });
}

// ---------- CHARTS (downsample p/ leveza) ----------
const MAX = msg.maxPoints ?? flow.get('chartMaxPoints') ?? 3000;
function downsample(pts, max = 3000) {
  if (pts.length <= max) return pts;
  const step = Math.ceil(pts.length / max);
  const out = [];
  for (let i = 0; i < pts.length; i += step) out.push(pts[i]);
  return out;
}
const chartMsgs = Object.entries(byGas).map(([gas, pts]) => ({
  topic: gas,
  payload: downsample(pts, MAX)
}));
```

Figura 18 - Código para as tabelas e para os medidores

Este nó, tendo preparado as *payloads*, envia-as para os nós que foram preparados para a visualização

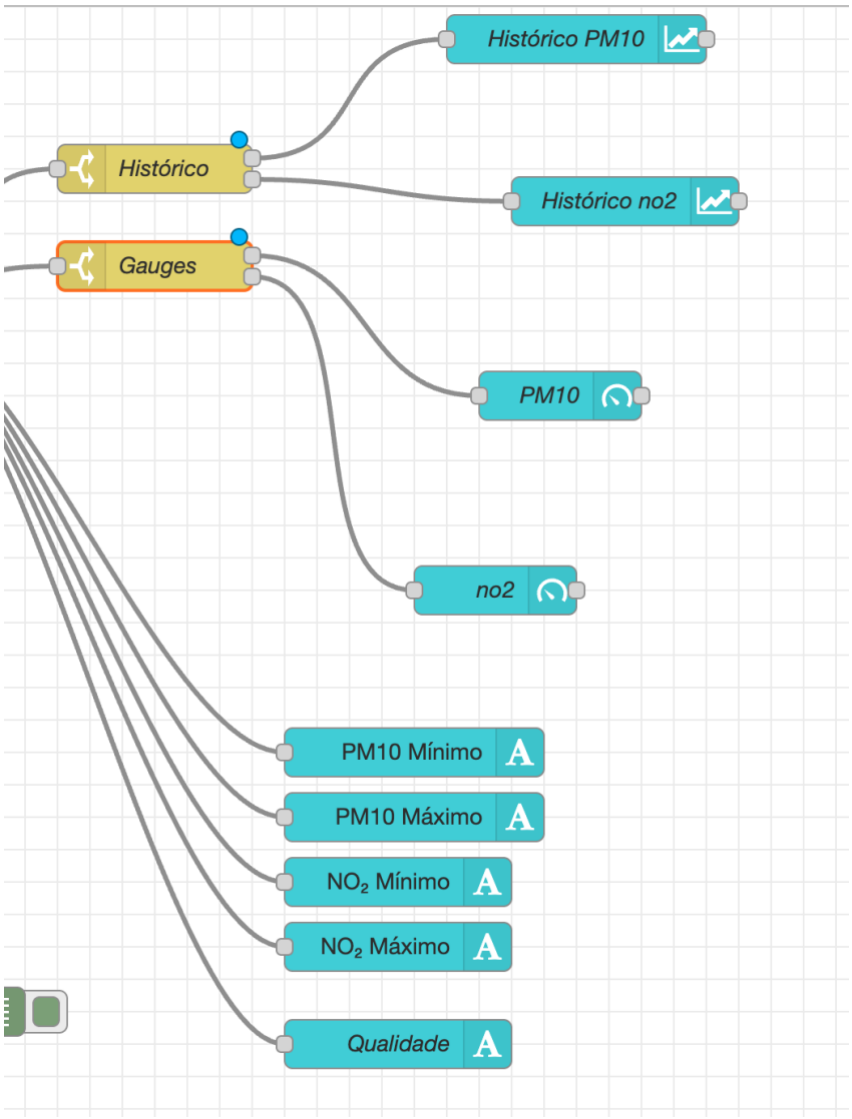


Figura 19 - Nós para visualização

Assim que estes Nós recebem os dados, o fluxo ETL está concluído e os dados estão preparados para visualização, no *dashboard* do Node-Red, geralmente localizado no *URL* "<ip.da.máquina:porta>/Dashboard".

5. Debug e Logs

Durante o processo *ETL* foram gerados diversos *Logs*, em diferentes partes do fluxo.

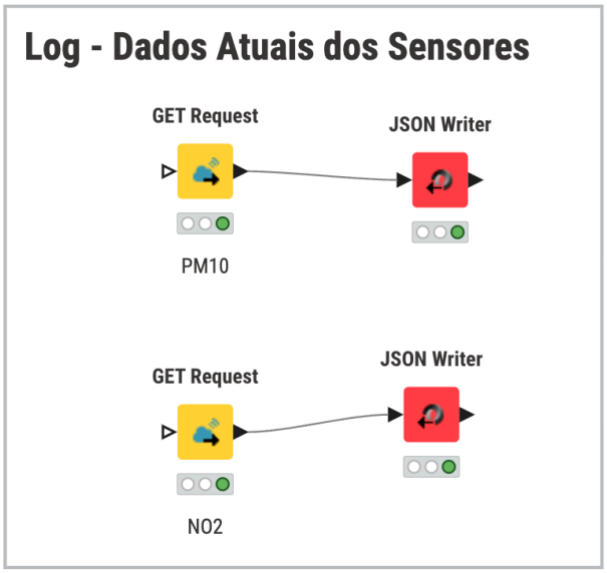


Figura 20 - Logs Dados Atuais

Para começar, estes nós são executados imediatamente ao começar o fluxo. Fazem pedidos à API para obter os dados mais recentes de cada um dos sensores. Isto é útil para confirmar que a API está online, e quando foi a última vez que o sensor mediu algo, assim como a última medição. Guardam um JSON com a resposta da API.

No final do fluxo onde obtemos os dados históricos de cada um dos sensores, também guardamos uma tabela CSV para termos os dados não tratados de cada um dos sensores.

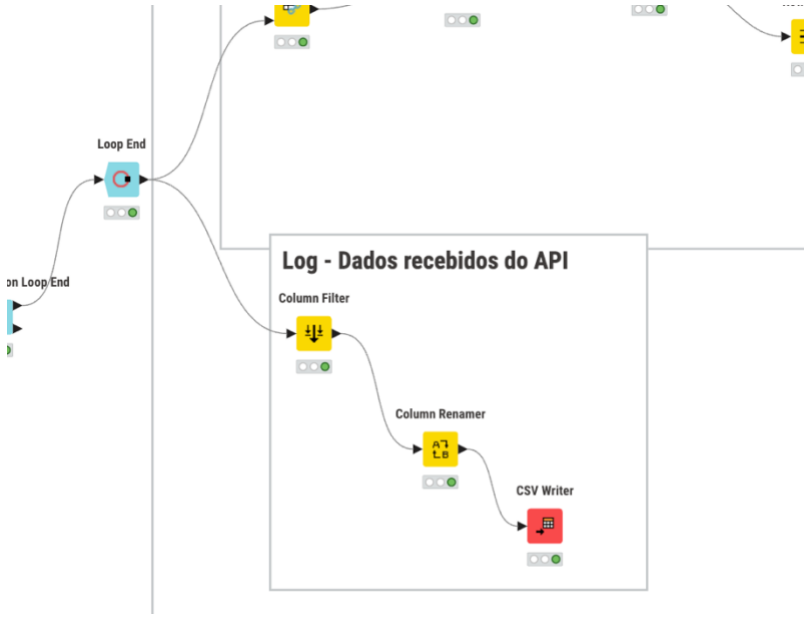


Figura 21 - Log dados não tratados

Após concatenar os dados entre os dois sensores e renomear algumas colunas, também criamos um log em JSON e em CSV, para verificar que temos tudo concatenado corretamente.

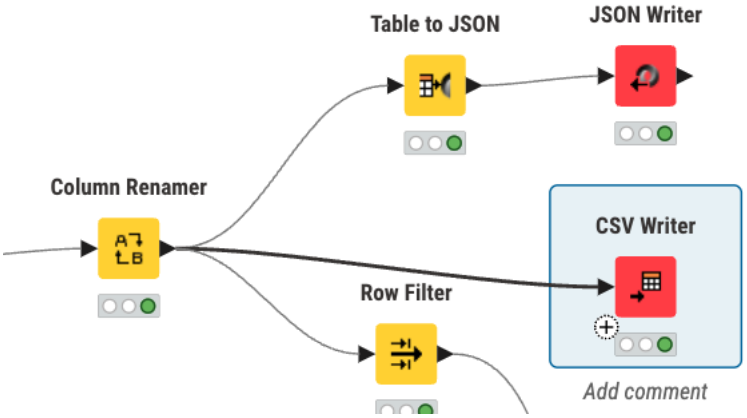


Figura 22 - Log depois de concatenar

Depois de fazer todo o tratamento de dados necessário, também criamos um Log, em formato CSV e JSON, dos dados que serão enviados para o Node-Red, para garantir que está tudo correto.

Figura 23 - Log dos dados tratados e prestes a enviar.

Tendo os dados enviados para o Node-Red, também é boa ideia ter Logs dos dados que foram recebidos do Knime. Para tal, um nó que imprima na janela de Debug o objeto JSON recebido foi a escolha tomada.

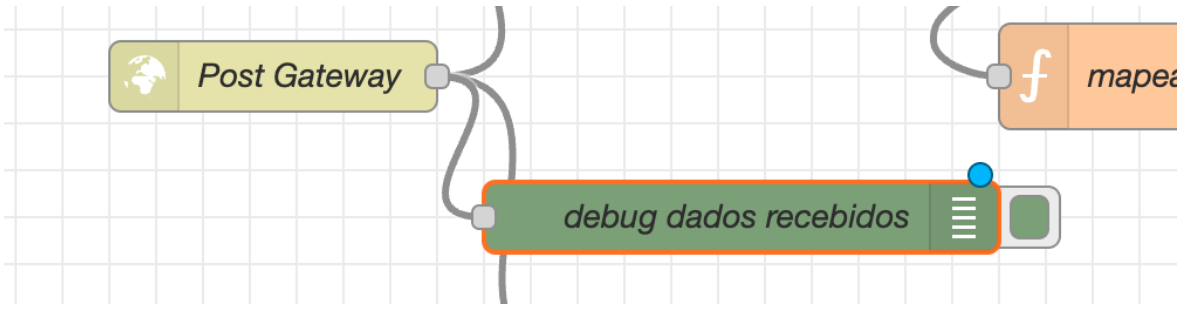


Figura 24- Imprimir JSON recebido

Este nó imprime em formato de Array o objeto que foi recebido.

Adicionalmente também existe um último nó de debug que imprime as coordenadas x e y do gráfico de dados históricos.

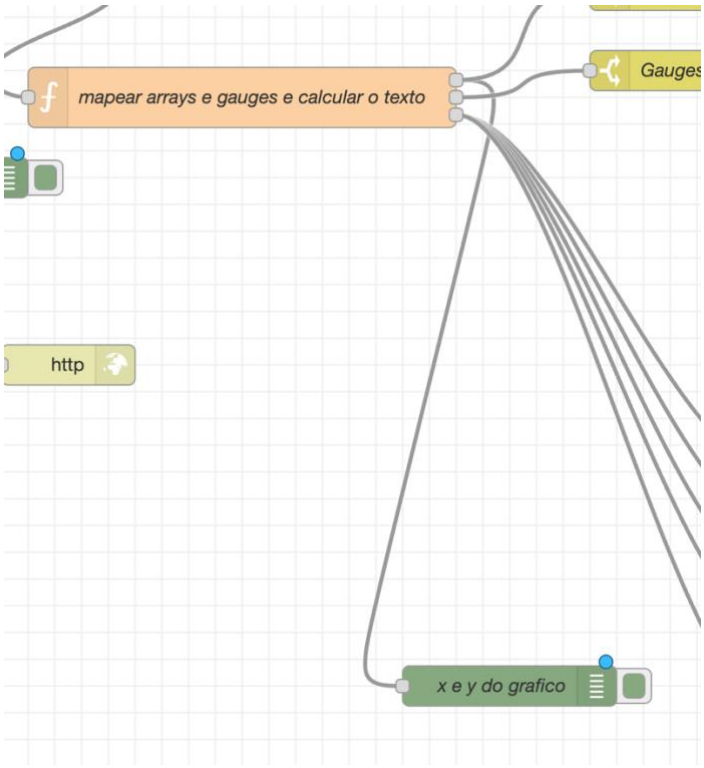


Figura 25 - debug grafico histórico

6. Projeto e Demonstração

6.1. Github

O repositório do projeto está disponível no *Github*, com instruções em como instalar e executar, no link a seguir.

https://github.com/AnyTng/ProjetoISI_1

6.2. Vídeo

Um vídeo de demonstração está disponível na plataforma *YouTube*, a partir do seguinte QR Code, ou link

Figura 26 - QR Code

https://www.youtube.com/watch?v=fiKXs676lkg

7. Conclusão

Este projeto serviu para consolidar conhecimentos de integração de sistemas de informação ao nível dos dados, implementando um processo de *ETL* (*Extract*, *Transform* e *Load*) para recolher, tratar e carregar medições de PM10 e NO_2 a partir da *API* do *OpenAQ*.

Com o *KNIME* orquestrei pedidos paginados, normalizei unidades e carimbos temporais, tratei valores ausentes (incluindo a identificação de lacunas em PM10 entre agosto e outubro de 2024), removi duplicados e leituras anómalas e produzi dados em *JSON* prontos para consumo. Em seguida, integrei com o *Node-RED* para construir uma *dashboard* simples que permite visualizar séries temporais e indicadores atuais por poluente. O trabalho evidenciou a importância da limpeza, normalização e agregação para garantir consistência analítica, bem como a utilidade de automatizar o fluxo desde a extração até à visualização. Como perspetiva futura, sugere-se alargar a ingestão a outros parâmetros (por exemplo, PM2.5 e O_3), adicionar validações de qualidade e notificações em tempo real, e explorar modelos preditivos leves para apoio à decisão.

8. Referências Bibliográficas

- OpenAQ API Documentation - https://docs.openaq.org/about/about
- OpenAQ API - https://api.openaq.org/docs
- European Air Index Calculation - https://ecmwf-projects.github.io/copernicus-training-cams/proc-aq-index.html
- European Air Quality Index - https://airindex.eea.europa.eu/AQI/index.html
- Knime Documentation - https://docs.knime.com/
- Node-Red Documentation - https://nodered.org/docs/