UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Nicolas Ignacio Ryberg

DESENVOLVIMENTO DE UM SISTEMA IOT DE MONITORAMENTO DA QUALIDADE DO AR PARA A STARTUP IRIS

Florianópolis
2024

Nicolas Ignacio Ryberg

DESENVOLVIMENTO DE UM SISTEMA IOT DE MONITORAMENTO DA QUALIDADE DO AR PARA A STARTUP IRIS

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Elétrica do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito parcial para a obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Leonardo Hoinaski, Dr.

Florianópolis
2024

Ryberg, Nicolas Ignacio
DESENVOLVIMENTO DE UM SISTEMA IOT DE MONITORAMENTO DA QUALIDADE DO AR PARA A STARTUP IRIS / Nicolas Ignacio Ryberg ; orientador, Leonardo Hoinaski, 2024.
118 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia Elétrica, Florianópolis, 2024.

Inclui referências.

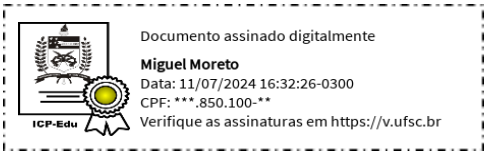
1. Engenharia Elétrica. 2. Monitoramento da Qualidade do Ar. 3. IoT. 4. Desenvolvimento de Sistemas. 5. Planejamento de Projeto. I. Hoinaski, Leonardo. II. Universidade Federal de Santa Catarina. Graduação em Engenharia Elétrica. III. Título.

Nicolas Ignacio Ryberg

DESENVOLVIMENTO DE UM SISTEMA IOT DE MONITORAMENTO DA QUALIDADE DO AR PARA A STARTUP IRIS

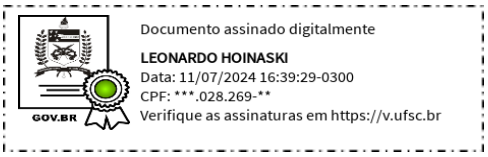
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Engenharia Elétrica e aprovado em sua forma final pelo Curso de Engenharia Elétrica

Florianópolis, 11 de julho de 2024.



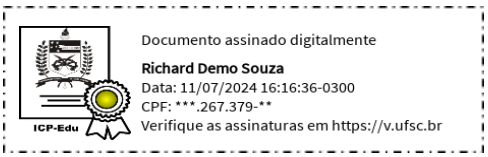
Coordenação do Curso

Banca examinadora



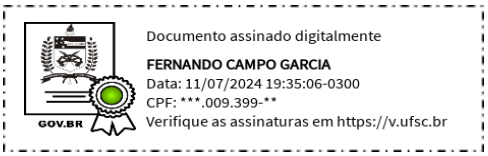
Prof. Leonardo Hoinaski, Dr.

Orientador(a)



Prof. Richard Demo Souza, Dr.

Instituição UFSC



Fernando Campo Garcia, Dr.

Florianópolis, 2024.

Dedico este Trabalho de Conclusão de Curso aos meus pais, Ronald e Florência, por serem meus exemplos de dedicação e sabedoria.

RESUMO

O interesse no monitoramento da qualidade do ar tem apresentado um crescimento acelerado nos últimos anos. Para suprir esse interesse, soluções de baixo custo têm surgido para aumentar a malha de monitoramento e complementar as medições das estações de referência. Algumas plataformas têm surgido para consolidar essas informações, mas estas ainda são esparsas e não atendem todos os requisitos demandados por uma solução comercial. Este trabalho descreve o desenvolvimento de um sistema que atenda a essa demanda, criado para a startup IRIS. O objetivo é criar um sistema eficiente para coleta, processamento e análise de dados ambientais, utilizando sensores de baixo custo. O trabalho envolve o levantamento dos requisitos do sistema, a construção de uma arquitetura robusta e o desenvolvimento deste. O sistema desenvolvido é escalável, flexível e capaz de integrar dados de fontes diversas, garantindo a precisão e a confiabilidade dos dados monitorados. O resultado é um sistema funcional, destacando-se por sua capacidade de calibração em tempo real e integração com estações de referência. Este sistema permite que a IRIS inicie operações com clientes reais, validando suas hipóteses de mercado e recebendo feedback para futuras melhorias.

Palavras-chave: IoT, Monitoramento da qualidade do ar, Sensores de baixo custo, Integração de dados, Desenvolvimento Web, Calibração em Tempo Real.

ABSTRACT

Interest in air quality monitoring has seen rapid growth in recent years. To meet this demand, low-cost solutions have emerged to expand monitoring networks and complement reference stations. Several platforms have been developed to consolidate this information, but they remain sparse and do not meet all commercial requirements. This work describes the development of a system for the startup IRIS, aimed at efficiently collecting, processing, and analyzing environmental data using low-cost sensors. The system is scalable, flexible, and capable of integrating data from various sources, ensuring data accuracy and reliability. This functional system features real-time calibration and integration with reference stations, allowing IRIS to operate with real clients, validate market hypotheses, and receive feedback for future improvements.

Keywords: IoT, Air Quality Monitoring, Low-Cost Sensors, Data Integration, Web Development, Real-Time Calibration.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALB	Application Load Balancer
AWS	Amazon Web Services
BI	Business Intelligence
CO	Monóxido de Carbono
CO2	Dióxido de Carbono
CONAMA	Conselho Nacional do Meio Ambiente
CRUD	Create, Read, Update, Delete
CSV	Comma-Separated Values
DB	Banco de Dados (Database)
DDD	Domain Driven Design
DTO	Data Transfer Object
ECS	Amazon Elastic Container Service
ETL	Extract, Transform, Load
GCP	Google Cloud Platform
GPRS	General Packet Radio Service
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IAM	Identity and Access Management
IA	Inteligência Artificial
IEMA	Instituto de Energia e Meio Ambiente
IoT	Internet das Coisas (Internet of Things)
IP	Internet Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
LoRaWAN	Long Range Wide Area Network
MMA	Ministério do Meio Ambiente
MQTT	Message Queuing Telemetry Transport
MVP	Produto Mínimo Viável (Minimum Viable Product)
OMS	Organização Mundial da Saúde
POC	Prova de Conceito (Proof of Concept)
PM2.5	Partículas Inaláveis com Diâmetro Menor ou Igual a 2,5 Micrômetros

PM10	Partículas Inaláveis com Diâmetro Menor ou Igual a 10 Micrômetros
REST	Representational State Transfer
RLS	Row Level Security
SPA	Single Page Application
SQL	Structured Query Language
TSDB	Time Series Database
TCC	Trabalho de Conclusão de Curso
UFSC	Universidade Federal de Santa Catarina
VM	Máquina Virtual (Virtual Machine)

SUMÁRIO

1 INTRODUÇÃO

Esse documento descreve parte do desenvolvimento de um sistema IoT de monitoramento da qualidade do ar. O sistema foi desenvolvido para a startup IRIS, que desenvolve equipamentos de monitoramento da qualidade do ar de baixo custo e presta serviços relacionados ao tema.

A poluição do ar é uma das maiores ameaças ambientais, sendo responsável por cerca de 7 milhões de mortes prematuras anuais, segundo a Organização Mundial da Saúde (OMS, 2021). Tradicionalmente, o monitoramento da qualidade do ar é realizado por estações fixas, que são caras e complexas de operar, limitando sua utilização em regiões menos desenvolvidas (Borrego et al., 2016). Com a incorporação de sensores portáteis, de baixo custo, acessíveis e de fácil utilização, o paradigma do monitoramento da qualidade do ar está mudando. Estes dispositivos fornecem dados com alta resolução temporal e espacial, permitindo uma análise mais detalhada e em tempo real das condições ambientais (Snyder et al., 2013).

Desta forma, a Internet das Coisas (IoT) surge como uma ferramenta poderosa para monitorar e mitigar esses impactos, utilizando uma tecnologia de rede integrada, sinérgica e onipresente para fornecer serviços inteligentes em várias áreas (Guo; Zhu; Yang, 2016). Esse aumento na malha de monitoramento causado pela acessibilidade dos sensores de baixo custo, exige a construção de uma solução IoT que consiga atender aos requisitos de escalabilidade do volume dos dados, tanto do ponto de vista de conectividade, processamento, armazenamento e análise (Iqbal, 2021).

Existem diferentes plataformas que buscam endereçar o problema de consolidar e disponibilizar dados de poluição do ar (Khot et al., 2018), algumas das quais serão apresentadas neste documento. Porém, essas plataformas não atendem as necessidades da IRIS por completo: gestão dos equipamentos da frota IoT; consolidação de dados de diferentes fontes; aquisição, processamento e calibração em tempo real das medidas; disponibilização dos dados e análises; e segurança e acesso controlado às informações.

Portanto, este trabalho visa desenvolver a solução de monitoramento da qualidade do ar da IRIS que atenda aos requisitos acima. Para isso, primeiro foi realizado um estudo de mercado, com o objetivo de levantar os requisitos da plataforma. Então, foi desenhada uma arquitetura que conseguisse atender aos requisitos. Por fim, foi priorizado o escopo para o desenvolvimento da primeira versão da plataforma. Espera-se que, com a implementação

dessa solução, seja possível começar a operar uma rede de monitoramento do ar em clientes finais.

1.1 OBJETIVOS

1.1.1 Objetivo geral

Desenvolvimento de um sistema para aquisição de dados, operação e análise de redes de monitoramento da qualidade do ar da startup IRIS que habilite a empresa a operar com potenciais clientes.

1.1.2 Objetivos específicos

Objetivo específico 1: Levantar os requisitos do sistema e planejar o que deverá ser desenvolvido para permitir que o sistema atenda as necessidades de curto, médio e longo prazo da empresa.

Objetivo específico 2: Montar uma arquitetura de tecnologia que atenda aos requisitos definidos como prioritários.

Objetivo específico 3: Desenvolver o sistema planejado nos objetivos anteriores.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 POLUIÇÃO ATMOSFÉRICA

A poluição do ar é uma das maiores ameaças ambientais à saúde humana e aos ecossistemas, causando cerca de 7 milhões de mortes prematuras anuais (WHO, 2021). As consequências da poluição do ar são vastas, afetando tanto a saúde humana quanto o meio ambiente. As principais causas da poluição do ar incluem a queima de combustíveis fósseis (como carvão, petróleo e gás), atividades industriais, emissões veiculares, desmatamento e queimadas, além de processos naturais como erupções vulcânicas e tempestades de poeira (Seinfeld & Pandis, 2016).

Alguns aspectos cruciais da poluição do ar incluem a identificação das fontes de poluentes, a caracterização dos tipos de poluentes (gases, partículas finas e ultrafinas, compostos orgânicos voláteis), e a compreensão dos efeitos desses poluentes na saúde e no meio ambiente. Outro aspecto importante é a variabilidade espacial e temporal da poluição, que pode ser influenciada por fatores como clima, topografia e padrões de uso do solo (Harrison et al., 2020).

A gestão da qualidade do ar refere-se ao conjunto de ações e políticas destinadas a controlar e reduzir a poluição do ar para proteger a saúde pública e o meio ambiente. Isso envolve o monitoramento contínuo da qualidade do ar, a regulamentação das emissões de poluentes, o desenvolvimento de estratégias de mitigação e a promoção de práticas sustentáveis (Kumar et al., 2015). Para ser eficaz, a gestão da qualidade do ar deve ser baseada em uma abordagem integrada que considere todas as fontes de poluição e suas interações (European Environment Agency, 2018).

O monitoramento da qualidade do ar é o primeiro passo essencial para uma gestão eficaz da poluição do ar. Ele fornece os dados necessários para avaliar os níveis de poluição, identificar fontes de emissão e entender as tendências e padrões de poluição ao longo do tempo. Além disso, o monitoramento contínuo permite a avaliação da eficácia das políticas de controle de poluição e a tomada de decisões informadas para a proteção da saúde pública e do meio ambiente (Pope et al., 2009).

O monitoramento atmosférico pode ser realizado por meio de estações de referência e estações de baixo custo. As estações de referência são altamente precisas e confiáveis, utilizando equipamentos avançados como cromatógrafos e espectrômetros de massa para medir a concentração de poluentes. No entanto, são onerosas e complexas de operar, o que

limita sua implantação em larga escala, especialmente em regiões menos desenvolvidas (Borrego et al., 2016). Por outro lado, as estações de baixo custo utilizam sensores portáteis e acessíveis que, embora menos precisos, podem ser distribuídos em maior quantidade para fornecer uma cobertura mais ampla e detalhada da variabilidade espacial e temporal da poluição do ar (Snyder et al., 2013).

2.2 O MONITORAMENTO DA QUALIDADE DO AR

O monitoramento da qualidade do ar consiste na medição contínua de poluentes atmosféricos para avaliar as concentrações e verificar se os níveis estão dentro dos padrões estabelecidos pelas autoridades ambientais. Diversas ferramentas são utilizadas no monitoramento, como as estações de referência e estações de baixo custo.

O monitoramento de referência é realizado por estações fixas altamente sofisticadas Estas estações são equipadas com instrumentos de alta precisão, como cromatógrafos e espectrômetros de massa, capazes de medir uma ampla gama de poluentes atmosféricos com grande acurácia. A principal vantagem dessas estações é a alta qualidade e confiabilidade dos dados, que servem como base para a formulação de políticas públicas e regulamentações ambientais. No entanto, as estações de referência têm desvantagens significativas, incluindo altos custos de instalação e manutenção, complexidade operacional e limitação na cobertura espacial devido ao número reduzido de estações disponíveis (Borrego et al., 2016).

Dada a limitação das estações de referência, especialmente em regiões menos desenvolvidas, há uma crescente demanda por alternativas de monitoramento que sejam mais acessíveis e ampliem a cobertura espacial. Uma dessas alternativas é o monitoramento de baixo custo, que utiliza sensores portáteis e acessíveis para medir poluentes atmosféricos. Estes sensores, embora não possuam a mesma precisão das estações de referência, oferecem uma solução viável para ampliar a rede de monitoramento ambiental, proporcionando uma maior resolução espacial dos dados (Kumar et al., 2015).

O monitoramento de baixo custo baseia-se em sensores de menor precisão, mas que são economicamente acessíveis e fáceis de usar. Esses sensores medem concentrações de poluentes de forma contínua e podem ser distribuídos em maior número, aumentando a cobertura espacial e permitindo uma análise mais detalhada da variabilidade dos poluentes em diferentes áreas. A aplicação desses sensores é vasta, incluindo áreas urbanas densamente povoadas, regiões rurais e locais próximos a fontes de poluição (Snyder et al., 2013).

Apesar das vantagens, o monitoramento de baixo custo enfrenta desafios significativos, como a precisão, durabilidade e confiabilidade dos sensores, que variam conforme a tecnologia de medição, componentes, condições ambientais e métodos de operação. A calibração dos sensores de baixo custo é fundamental para garantir a qualidade dos dados. Este processo envolve ajustar as leituras dos sensores para que correspondam às medições realizadas pelas estações de referência, utilizando técnicas de aprendizado de máquina e regressões multivariadas (Harrison et al., 2020).

A combinação de estações de referência com sensores de baixo custo calibrados em tempo real oferece uma solução robusta para o monitoramento da qualidade do ar. As estações de referência fornecem dados de alta qualidade que podem ser usados para calibrar e validar os sensores de baixo custo. Esta integração permite uma maior cobertura espacial e temporal, melhorando a capacidade de detecção de hotspots de poluição e a análise de tendências de longo prazo (European Environment Agency, 2018).

Os maiores desafios no campo do monitoramento de baixo custo incluem a calibração contínua dos sensores, a integração de dados de diferentes fontes e a garantia de precisão e confiabilidade em diversas condições ambientais. Além disso, há uma necessidade constante de desenvolvimento de novas tecnologias e metodologias para melhorar a qualidade dos dados coletados. Campos de estudo emergentes incluem a aplicação de algoritmos de aprendizado de máquina para calibração em tempo real, desenvolvimento de novos materiais para sensores e a criação de redes integradas de monitoramento ambiental (Gubbi et al., 2013).

2.3 PLATAFORMAS DE MONITORAMENTO DA QUALIDADE DO AR

As plataformas de monitoramento da qualidade do ar oferecem diferentes abordagens e tecnologias para a coleta de dados de poluentes. Entre as diversas plataformas disponíveis, destacam-se MonitorAr, IEMA (Instituto de Energia e Meio Ambiente), PurpleAir, OpenAQ e Kunak. Cada uma dessas plataformas e ferramentas oferece uma abordagem única para o monitoramento da qualidade do ar, contribuindo de maneiras específicas para a coleta e análise de dados sobre poluentes atmosféricos.

O monitoramento em tempo real, a alta resolução espacial dos dados e a integração de tecnologias de baixo custo são algumas das principais vantagens dessas plataformas. Além

disso, a calibração dos sensores e a validação dos dados são aspectos cruciais para garantir a qualidade e a precisão das medições.

Essas plataformas foram estudadas nos seguintes critérios: integração de fontes de dados, conectividade dos equipamentos da rede, técnicas de processamento das medidas, modelagem da tecnologia e dashboards de análise.

2.3.1 MonitorAr

A plataforma MonitorAr é uma iniciativa brasileira desenvolvida pelo Ministério do Meio Ambiente (MMA) para monitorar a qualidade do ar em tempo real em várias regiões do Brasil. Ela fornece dados sobre a concentração de poluentes atmosféricos, como material particulado, ozônio e dióxido de nitrogênio, permitindo uma análise detalhada da qualidade do ar. A plataforma também gera relatórios periódicos que auxiliam na formulação de políticas públicas e na implementação de ações para melhorar a qualidade do ar (MMA, 2024).

Integração de Fontes de Dados: Os dados disponibilizados no MonitorAr são provenientes das redes de monitoramento da qualidade do ar geridas pelas Unidades da Federação (UF). Essas informações são compiladas, analisadas e armazenadas na plataforma digital nacional única, hospedada pelo MMA, por meio da qual é possível consultar informações atualizadas da qualidade do ar nacionalmente (MMA, 2024). Atualmente, 12 Unidades da Federação (UF) possuem monitoramento da qualidade do ar em seus territórios, quais sejam: Bahia, Ceará, Espírito Santo, Mato Grosso do Sul, Minas Gerais, Paraná, Pernambuco, Rio de Janeiro, Rio Grande do Sul e São Paulo. Além dessas, cabe destacar que o Distrito Federal e Goiás realizam o monitoramento de poluentes de forma manual, com isso, a geração de dados não é tão rápida e eficaz quanto pelas redes automáticas (MonitorAr, 2024). Não foi possível encontrar como essa integração é realizada, se as fontes de dados possuem uma API para integrar as medidas, ou se essa ingestão é orquestrada pela MonitorAr.

Conectividade dos Equipamentos: Não se aplica. A plataforma não opera redes de monitoramento, apenas integra bases de dados de diferentes fontes.

Processamento das Medidas: Como a plataforma serve como uma base de integração, as medidas de concentração final já estão processadas no momento da ingestão dos dados. A plataforma, no entanto, calcula o Índice de Qualidade do Ar (IQAr), criado para facilitar a divulgação dos dados de monitoramento da qualidade do ar de curto prazo, conforme estabelecido pela Resolução Conama nº 491/18, tornando mais fácil o entendimento dos resultados pela população. Já para fins de gestão da qualidade do ar, os técnicos especializados analisam as concentrações de poluentes obtidas no monitoramento (Guia Técnico MMA, 2023). O IQAr é calculado conforme apresentado na figura 1.

Figura 1: Cálculo do Índice da Qualidade do Ar

À concentração medida de cada poluente é atribuído um IQAr a partir da equação:

$$IQAr = I_{ini} + \frac{I_{fin} - I_{ini}}{C_{fin} - C_{ini}} \times (C - C_{ini})$$

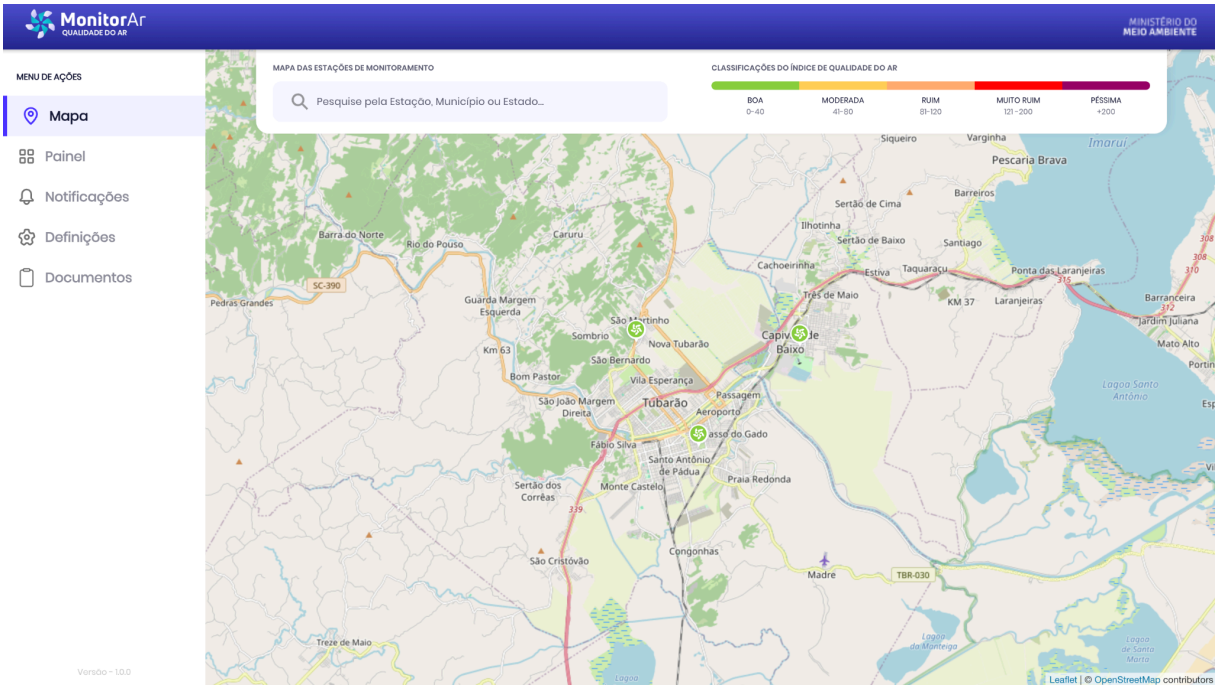
Equação 1

I_{ini} = valor do índice que corresponde à concentração inicial da faixa;
I_{fin} = valor do índice que corresponde à concentração final da faixa;
C_{ini} = concentração inicial da faixa em que se localiza a concentração medida;
C_{fin} = concentração final da faixa em que se localiza a concentração medida;
C = concentração medida do poluente.

Fonte: MMA (2024)

Dashboards de Análise: segue o padrão de uma tela de mapa e uma tela de detalhe.

Figura 2: Mapa de estações disponibilizado pelo MonitorAr



Fonte: MMA (2024)

Figura 3: Visão detalhada de estação disponibilizada pelo MonitorAr

Fonte: MMA (2024)

2.3.2 PurpleAir

A plataforma PurpleAir utiliza sensores de baixo custo para monitoramento da qualidade do ar em tempo real. Esses sensores medem a concentração de partículas finas (PM2.5 e PM10) e outros poluentes, e os dados são disponibilizados em mapas interativos acessíveis ao público. A principal contribuição da PurpleAir é a democratização do monitoramento da qualidade do ar, permitindo que indivíduos e comunidades instalem sensores em locais estratégicos, aumentando a resolução espacial dos dados coletados. Todas as informações desta seção foram referenciadas do site da PurpleAir (PurpleAir, 2024).

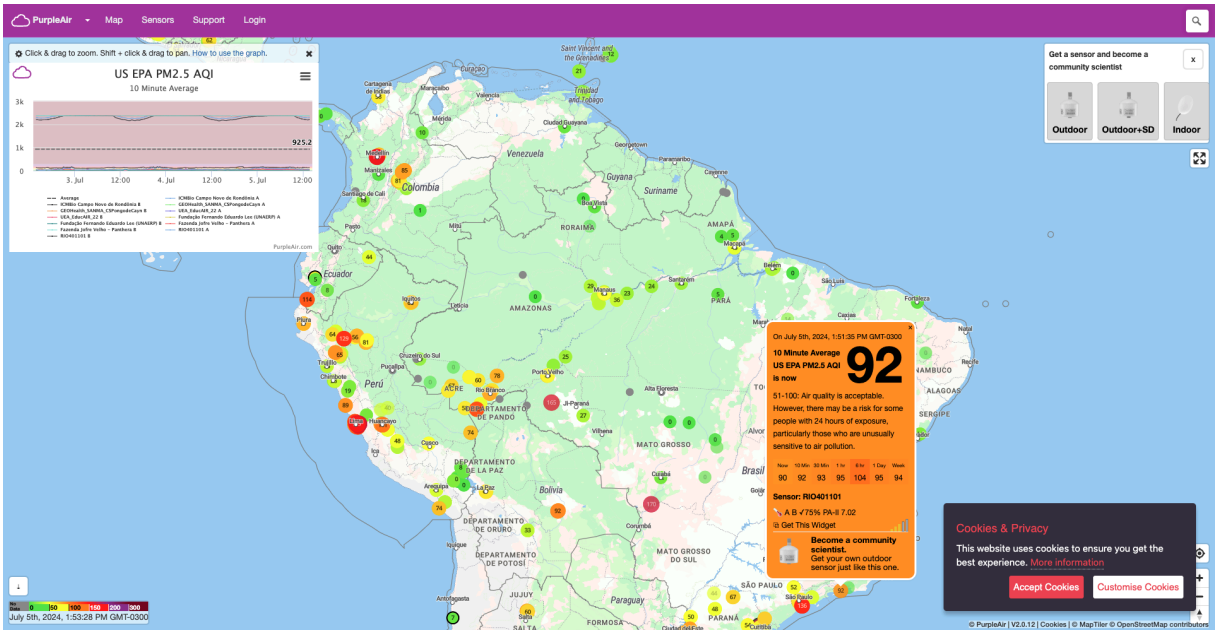
Integração de Fontes de Dados: Integra apenas os dados da própria frota.

Conectividade dos Equipamentos: Wi-Fi e resiliência de medidas por MicroSD.

Processamento das Medidas: O equipamento envia ao servidor dados processados de concentração onde é calculado de um índice da qualidade do ar

Dashboards de Análise: segue o padrão de uma tela de mapa e uma tela de detalhe.

Figura 4: Mapa de estações disponibilizado pelo PurpleAir



Fonte: PurpleAir (2024)

2.3.3 OpenAQ

A OpenAQ é uma plataforma de código aberto que agrega dados de qualidade do ar de diversas fontes ao redor do mundo. Ela coleta, armazena e disponibiliza dados de forma aberta, permitindo que pesquisadores, desenvolvedores e cidadãos acessem informações históricas e em tempo real sobre poluentes atmosféricos. A OpenAQ promove a transparência e o uso de dados abertos para a formulação de políticas públicas e pesquisas, contribuindo para uma maior compreensão global da qualidade do ar. Todas as informações desta seção foram referenciadas do site da OpenAQ (OpenAQ, 2024).

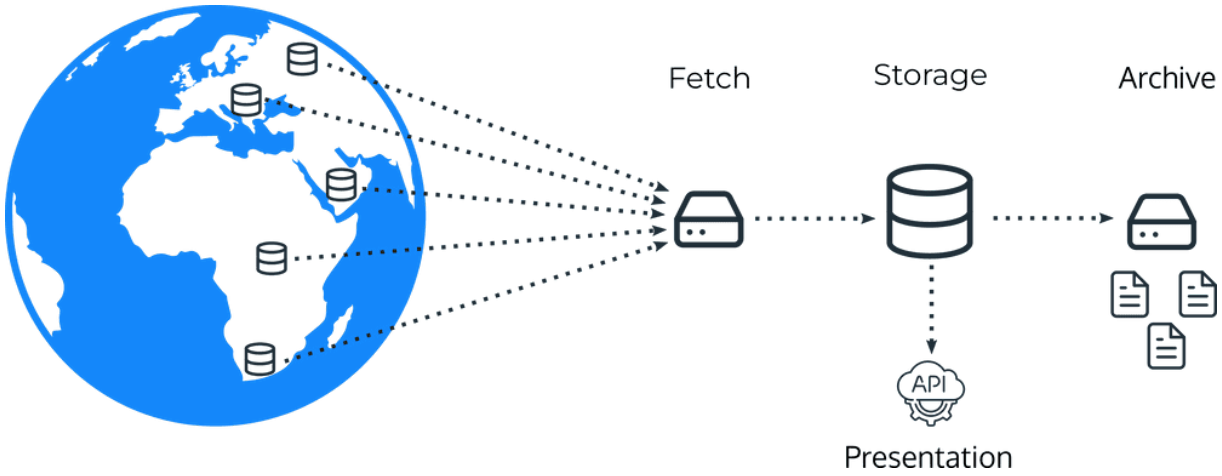
Integração de Fontes de Dados: Consegue integrar dados de qualquer fonte de informação. O processo de extração, transformação e carga de dados (ETL) flexível é descrito no item Modelagem da Tecnologia.

Conectividade dos Equipamentos: Wi-Fi e resiliência de medidas por MicroSD.

Processamento das Medidas: Não possuem frota proprietária.

Modelagem da Tecnologia: Uma vez que a OpenAQ atua como um integrador de dados, a tecnologia desenvolvida por eles é um processo de ETLs flexível para permitir integrar dados de diversas fontes em um padrão único. A ETL funciona em três etapas, como mostrado na figura 5.

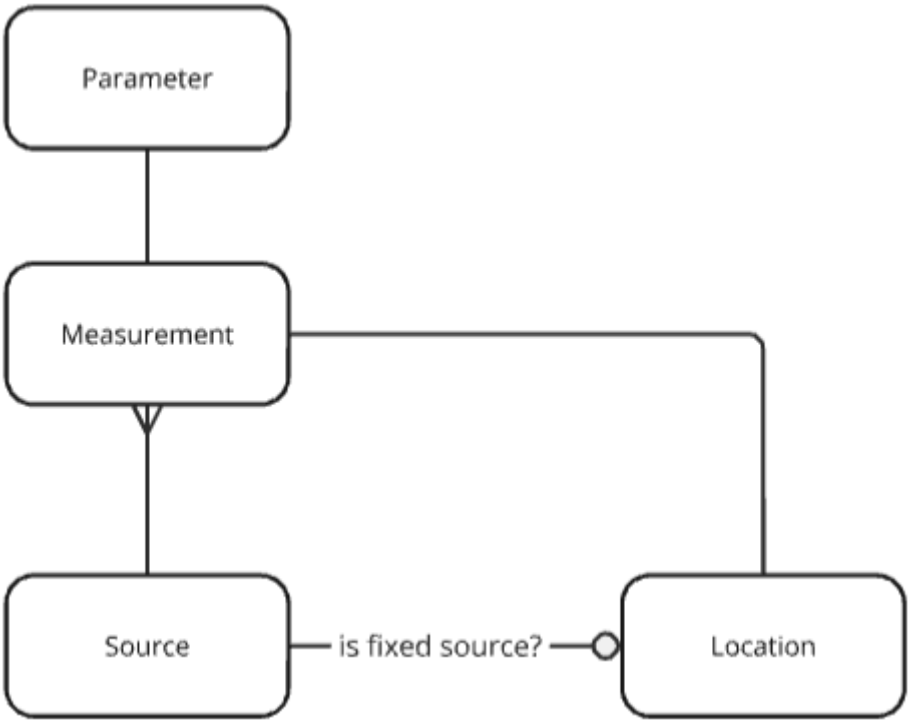
Figura 5: Arquitetura de ETL OpenAQ



Fonte: OpenAQ (2024)

- Fetch: adaptadores de captura são desenvolvidos para extrair dados de diversas fontes usando scripts que variam de raspadores HTML a scanners FTP e REST APIs. Repositórios principais: OpenAQ Fetch para fontes governamentais (NodeJS/Fargate); OpenAQ LCS Fetch para sensores de ar (NodeJS/Lambda Processes).
- Storage: os dados são armazenados em um banco de dados PostgreSQL com extensões TimeScale e PostGIS para funcionalidades temporais e geoespaciais. Código fonte: OpenAQ Database. Uma API REST é disponibilizada para acesso programático aos dados na camada de armazenamento.
- Archive: os dados são armazenados em um bucket público do S3.

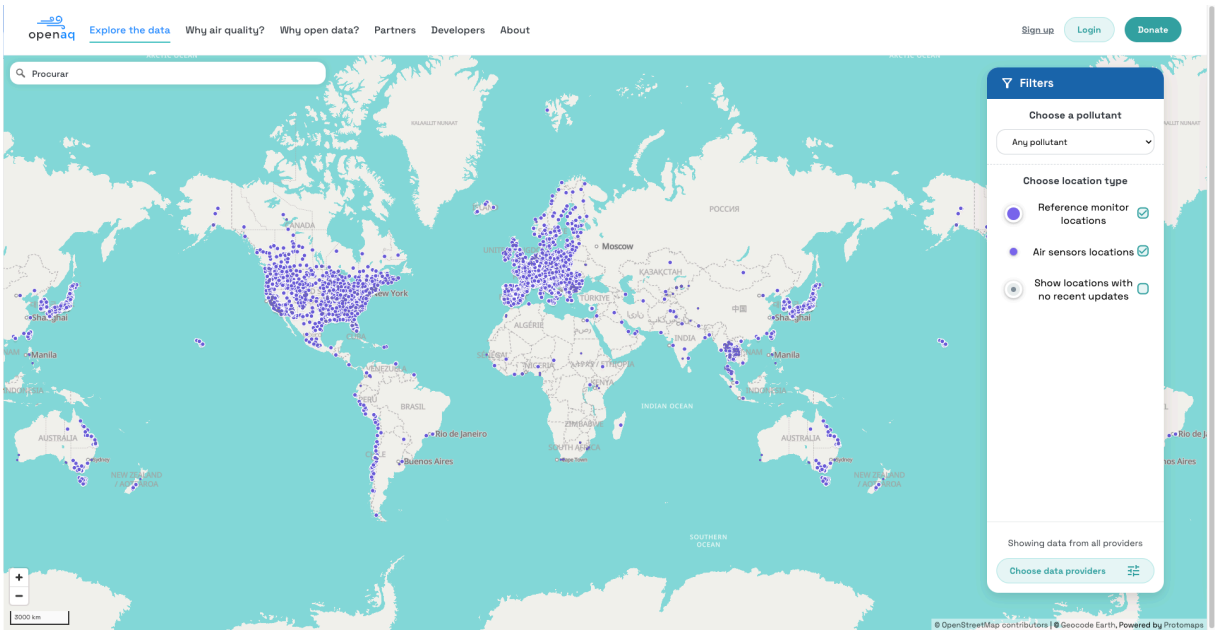
Figura 6: Modelagem das Entidades na Arquitetura da OpenAQ.



Fonte: Elaborado pelo autor (2024)

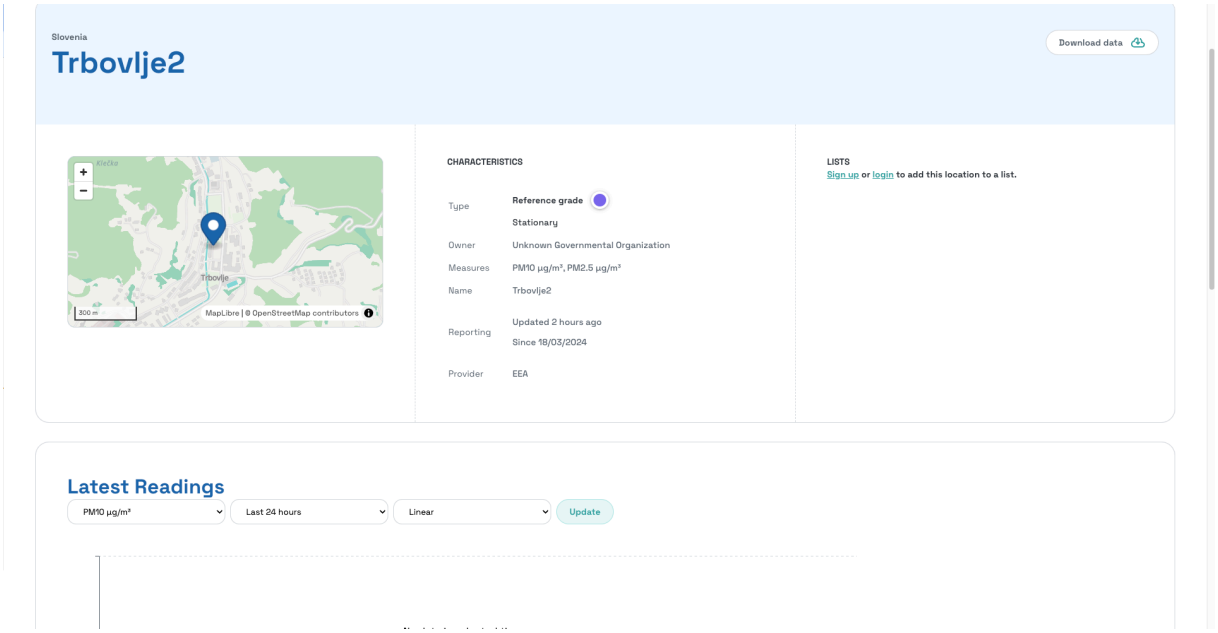
Dashboards de Análise: segue o padrão de uma tela de mapa e uma tela de detalhe.

Figura 7: Mapa de estações disponibilizado pela OpenAQ



Fonte: OpenAQ (2024)

Figura 8: Visão detalhada de estação disponibilizada pela OpenAQ



Fonte: OpenAQ (2024)

2.3.4 Kunak

A Kunak é uma das empresas líderes do mercado de monitoramento de baixo custo.

Integração de Fontes de Dados: Integra apenas os dados da própria frota.

Conectividade dos Equipamentos: Possui apenas conectividade 2G/3G/4G. Sem a possibilidade de conectividade por internet satelital / LoRaWAN.

Figura 9: Datasheet Kunak Pro

Specifications

Dimensions	257 x 270 x 225 mm
Weight	< 3.5 kg
Enclosure	PMMA & Polycarbonate & Stainless steel
Operating temp.	-20 ºC to 60ºC
Operating RH	0 to 99 %RH
IP rating	IP65
Battery	Lithium 26 Ah
External supply	7 - 12 Vdc. charger or solar panel
Autonomy	24/7 with charger or solar panel 9-30 days operation on battery (depending on configuration)
Power consumption	0.08 - 1.2W (depending on configuration)
Communications	Multi-Band 2G/3G/4G, Ethernet and Modbus RTU Slave
GNSS	GPS and GLONASS

Gas sensors	CO, CO_2, NO, NO_2, O_3, SO_2, H_2S, NH_3, CH_4, VOC, HCl
PM sensor	PM_1, $PM_{2.5}$, PM_4, PM_{10} TSP and TPC
Internal status	Temperature, battery, charging voltage and current, and signal
Built-in sensors	Temperature, humidity, atmospheric pressure and dew point
Connectors	#1: Power 7V to 12V or Ethernet #2: Modbus RTU Slave #3: Sound meter, UV #4: WBGT, Pyranometer, Modbus RTU Master #5: Anemometer & Rain Gauge
Sampling freq.	3Hz gases, 0.25Hz particles
Avg. periods	From 10 seconds to a maximum of 24 hours
Sending periods	From 5 minutes to a maximum of 24 hours
Remote management	Bidirectional communications Remote configuration and calibration
SIM	Embedded eSIM and SIM extra holder

Fonte: Kunak (2024)

Processamento das Medidas: A empresa não abre informações técnicas sobre o processamento dos dados, apenas que é realizada uma etapa de calibração e testes de acordo com a Classe 1 do padrão europeu CEN/TS 17660 (Kunak, 2024).

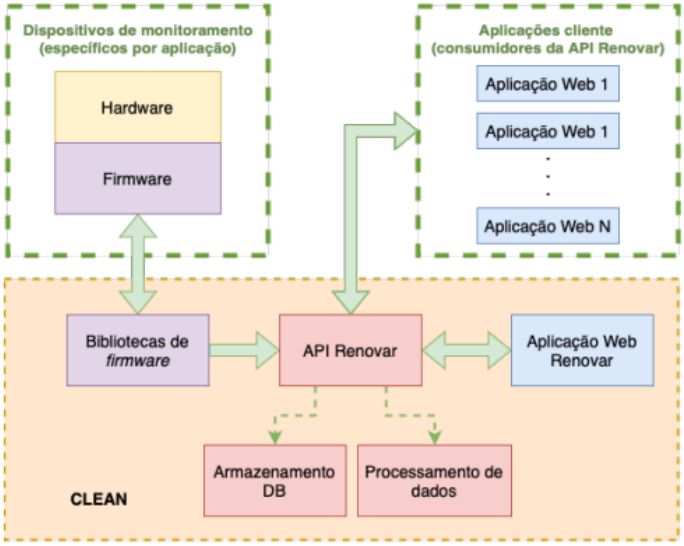
Dashboards de Análise: O sistema da empresa não é aberto ao público.

2.4 A INICIATIVA CLEAN – COLLABORATIVE LOW-COST ENVIRONMENTAL AIR QUALITY NETWORK

O presente trabalho é construído em cima da base criada pela iniciativa Collaborative Low-cost Environmental Air-quality Network (CLEAN), focando em realizar ajustes e adaptações necessárias para as aplicações comerciais da IRIS.

A iniciativa CLEAN tem como objetivo promover e facilitar o desenvolvimento de monitores de qualidade do ar de baixo custo por meio de uma plataforma colaborativa (Campo, 2023). A iniciativa é composta por quatro elementos principais: i) dispositivos de hardware, ii) firmware reutilizável, iii) guias e documentação para a reprodução do hardware e adesão à rede, e iv) a Plataforma Web Renovar para visualização e acesso remoto de dados em tempo real. Essas componentes são ilustradas graficamente na figura 10.

Figura 10 - Componentes da iniciativa CLEAN



Fonte: Campo (2023)

A iniciativa pode ser analisada sob a ótica dos seguintes itens:

Alimentação: A alimentação do dispositivo é fornecida pela rede elétrica, embora haja a opção de utilizar baterias em situações de falta de energia ou quando o acesso à rede elétrica é limitado. Essa flexibilidade permite que o dispositivo mantenha seu funcionamento mesmo em condições adversas.

Comunicação: Quanto à conectividade, o sistema implementa um módulo GPRS e Wi-Fi. O envio dos dados para o servidor é feito através de uma requisição HTTP em formato JSON (Campo, 2023).

Figura 11 - Componentes eletrônicos de comunicação dos dispositivos Clean

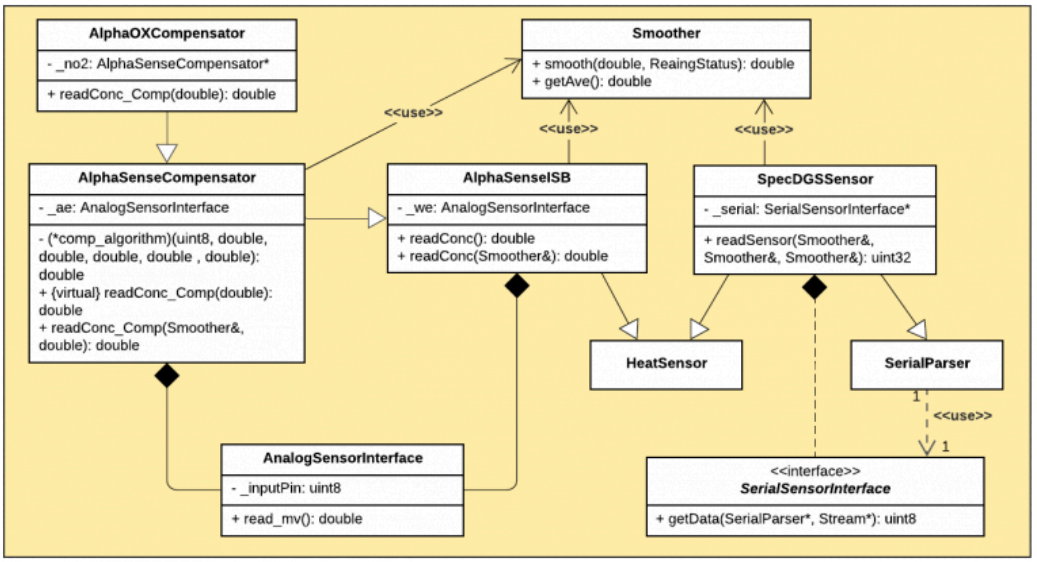
Cartão MicroSD	Cartão Micro SD Classe 10 de 16 GB	microSDHC SanDisk Ultra, da SanDisk
Módulo GPS	Módulo GPS NEO-6M c/ antena	GY-GPS6MV2, por u-blox
Módulo GPRS	Shield Arduino - GSM GPRS SIM900 com antena Quad Band	SIM900, da SIMCom
Módulo Wi-Fi	Módulo serial Wi-Fi ESP-01 ESP8266	ESP8266, da Expressif

Fonte: Campo (2023).

Processamento das medidas: Os algoritmos de compensação são aplicados pela classe AlphaSenseCompensator (Campo, 2023) pelas funções e métodos:

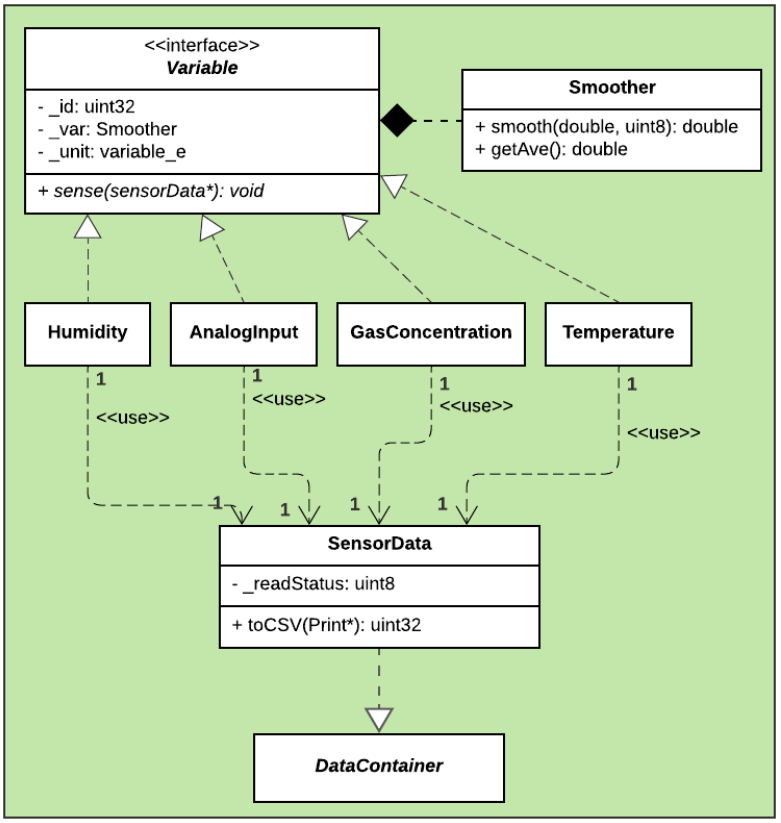
- (*comp_algorithm)(uint8, double, double, double, double, double): double: Este é um ponteiro para a função que implementa o algoritmo de compensação. As funções recebem como parâmetros as variáveis necessárias para o cálculo do algoritmo, dentre eles a temperatura.
- virtual readConc_Comp(double): double readConc_Comp(Smoother&, double): double Estes são métodos públicos que leem os valores de tensão armazenados nos atributos _we (herdados do AlphaSenseISB) e _ae. Eles aplicam o algoritmo de compensação correspondente e retornam um valor de concentração. Ambos os métodos recebem como parâmetros a temperatura ambiente e uma referência a um objeto Smoother, como no AlphaSenseISB.

Figura 12 - Diagrama de classes do pacote “Sensors”



Fonte: Campo (2023)

Figura 13 - Diagrama de classes do pacote “Data”

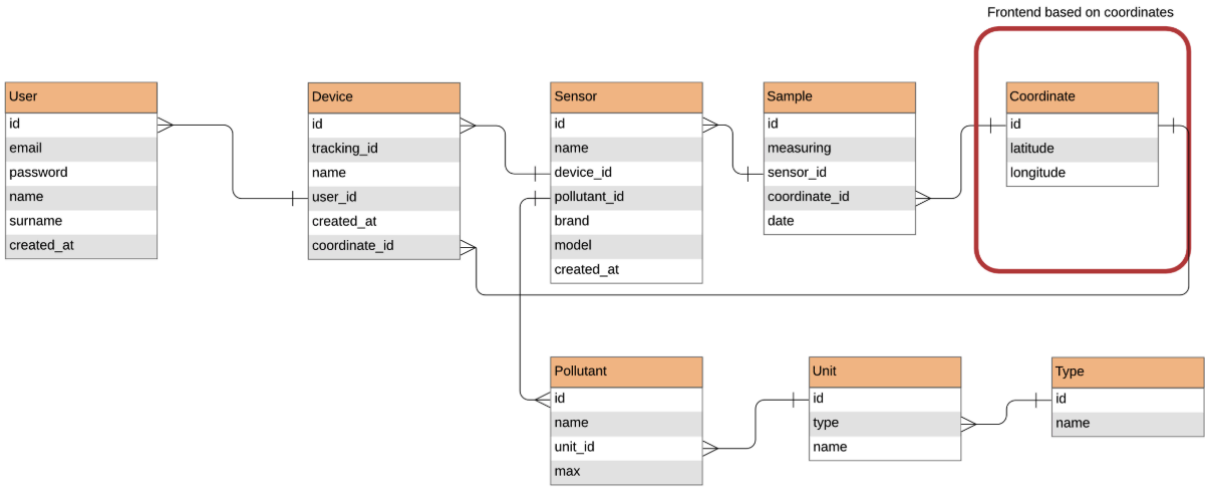


Fonte: Campo (2023)

Plataforma web Renovar: O backend implementa uma API REST consumida pelo equipamento que serve como uma interface de CRUD direta no banco, sem a aplicação de

regras de negócio significativas. O Banco de dados segue o seguinte modelo de dados mostrado na figura 14 e é implementado em MySQL:

Figura 14 - Modelo de dados da plataforma Renovar



Fonte: Campo (2023)

Também são implementadas algumas APIs de consumo dos dados, responsáveis por calcular agregações e devolver ao frontend dados formatados.

2.5 TECNOLOGIA IOT

A arquitetura de sistemas IoT é composta por várias camadas interconectadas que facilitam a coleta, transmissão, processamento e análise de dados. A primeira camada é a dos dispositivos físicos, que incluem sensores e atuadores responsáveis por coletar dados ambientais ou realizar ações específicas (Gubbi et al., 2013). A segunda camada é a de conectividade, que garante a transmissão de dados entre dispositivos e a plataforma central (Atzori et al., 2010). A terceira camada é das plataformas de processamento de dados, onde os dados coletados são armazenados, processados e analisados (Chen et al., 2014). Essa arquitetura é crucial para garantir a eficiência e eficácia do sistema IoT, permitindo uma resposta rápida e informada às condições monitoradas.

Algumas abordagens recentes criam mais divisões para as responsabilidades dessas três camadas. Um conceito interessante para a aplicação do trabalho é o conceito de Camada de Middleware: Gerencia a heterogeneidade dos dispositivos e fornece serviços essenciais, como gerenciamento de dispositivos, gerenciamento de dados e segurança, garantindo

interoperabilidade e integração contínua (Maksimović, 2020). Também a Camada de Borda: Fornece processamento preliminar de dados próximo à fonte, reduzindo a latência e o uso de largura de banda através de dispositivos de computação em borda (Shi et al., 2016).

Conectividade: A conectividade em sistemas IoT envolve a comunicação entre dispositivos e plataformas centrais, utilizando diversas tecnologias de rede. As opções incluem Wi-Fi, Bluetooth, Zigbee, LoRaWAN, e NB-IoT, cada uma adequada a diferentes tipos de aplicações e requisitos de alcance, consumo de energia e largura de banda (Raza et al., 2017). Essas tecnologias garantem que os dados coletados pelos sensores sejam transmitidos de forma confiável e eficiente para os sistemas centrais para processamento e análise (Sadeghi et al., 2015).

Segurança: A segurança é um aspecto crucial em sistemas IoT devido à natureza sensível dos dados e ao potencial de ataques cibernéticos. A autenticação e autorização são fundamentais para garantir que apenas usuários e dispositivos autorizados possam acessar o sistema.

- Autenticação e Autorização: Métodos comuns de autenticação e autorização em IoT incluem o uso de certificados digitais e JSON Web Tokens (JWT). Certificados digitais são usados para autenticar dispositivos, assegurando que eles são genuínos e confiáveis (Suo et al., 2012). JWTs são usados para gerenciar sessões de usuários, permitindo uma autenticação segura e eficiente (Jones et al., 2015).
- Segurança de Dados: Para garantir a segurança e integridade dos dados coletados e processados, técnicas como criptografia de dados em trânsito e em repouso são utilizadas (Sicari et al., 2015). Além disso, a implementação de firewalls, sistemas de detecção de intrusão e políticas de segurança robustas são essenciais para proteger a infraestrutura IoT contra ameaças (Roman et al., 2013).

Armazenamento de grandes volumes de dados: O armazenamento eficiente de grandes volumes de dados é um desafio significativo em sistemas IoT. Bancos de dados especializados, como Time Series Databases (TSDBs), são frequentemente utilizados para gerenciar dados de séries temporais gerados por dispositivos IoT (Palpanas, 2015). Ferramentas como TimescaleDB permitem a compressão, particionamento automático e otimizações específicas para consultas temporais, facilitando a gestão de grandes volumes de dados (Timescale, 2021). Além disso, tecnologias de armazenamento em nuvem, como

Amazon S3 e Google Cloud Storage, oferecem escalabilidade e resiliência, garantindo que os dados estejam sempre disponíveis e seguros (Gubbi et al., 2013).

Processamento dos dados: O processamento de dados em sistemas IoT envolve a transformação dos dados brutos coletados pelos sensores em informações úteis e acionáveis. "O processamento eficiente e o armazenamento de dados são fundamentais em infraestruturas IoT para apoiar a tomada de decisões em tempo real e o armazenamento a longo prazo" (Hanes et al., 2017). Técnicas de análise de dados em tempo real são frequentemente empregadas para monitorar condições e detectar anomalias imediatamente (Bonomi et al., 2012). Ferramentas de Big Data, como Apache Kafka e Apache Spark, são usadas para ingestão, processamento e análise de grandes volumes de dados IoT, permitindo uma resposta rápida às condições monitoradas (Kreps et al., 2011). Além disso, algoritmos de Machine Learning são aplicados para prever tendências e melhorar a precisão dos dados coletados (Zhang et al., 2015).

2.6 DESENVOLVIMENTO DE SOFTWARE

2.6.1 Princípios de arquitetura de software

- **Clean Architecture**: Esta abordagem, proposta por Robert C. Martin, enfatiza a separação de interesses e a criação de sistemas modulares, onde a lógica de negócio é independente de frameworks, bancos de dados ou interfaces de usuário. Isso facilita a manutenção e escalabilidade do software (Martin, 2017).
- **Domain Driven Design (DDD)**: Proposto por Eric Evans, o DDD foca na modelagem de domínios complexos com base na colaboração contínua entre especialistas no domínio e desenvolvedores. Isso ajuda a criar um software que reflita fielmente as regras de negócio e as necessidades dos usuários (Evans, 2003).
- **Princípios SOLID**: Os princípios SOLID são um conjunto de diretrizes de design de software que promovem a criação de sistemas mais compreensíveis, flexíveis e fáceis de manter. Esses princípios incluem a responsabilidade única, o aberto/fechado, a substituição de Liskov, a segregação de interfaces e a inversão de dependências (Martin, 2002).

2.6.2 Metodologia Ágil

A metodologia ágil, como o Scrum, promove ciclos de desenvolvimento curtos e iterativos, permitindo ajustes rápidos e entregas frequentes de valor ao cliente. Isso é essencial para projetos IoT, onde as necessidades podem evoluir rapidamente (Beck et al., 2001).

2.6.3 Cloud Computing

A utilização de serviços em nuvem, como Amazon Web Services (AWS) e Google Cloud, oferece escalabilidade, flexibilidade e resiliência, permitindo que a infraestrutura cresça conforme necessário sem grandes investimentos iniciais (AWS, 2023; Google Cloud, 2023).

A escalabilidade horizontal, ou escalonamento, é fundamental para lidar com o aumento de carga e de usuários em sistemas IoT. Ela envolve a adição de mais máquinas ou instâncias ao sistema para distribuir a carga de trabalho de maneira eficiente. Ferramentas como Kubernetes e Docker são frequentemente usadas para orquestrar e gerenciar contêineres em um ambiente escalável (Kubernetes, 2023; Docker, 2023). Construído em cima dessas ferramentas, o Amazon Elastic Container Service (ECS) é uma solução de orquestração de contêineres que simplifica a execução e a escalabilidade de aplicativos Docker na AWS. O ECS permite gerenciar clusters de contêineres e executar tarefas de maneira eficiente, integrando-se profundamente com outros serviços da AWS, como Identity and Access Management (IAM), CloudWatch, e Application Load Balancer (ALB), para fornecer segurança, monitoramento e balanceamento de carga automáticos (AWS, 2023).

2.6.4 Desenvolvimento Web

A arquitetura cliente-servidor é um modelo de design fundamental em aplicações web, onde as funções de cliente e servidor são separadas. O cliente é responsável por apresentar a interface do usuário e capturar as interações do usuário, geralmente através de navegadores web ou aplicativos móveis. O servidor, por outro lado, é responsável por fornecer serviços, como armazenamento de dados, processamento de lógica de negócios e resposta às solicitações dos clientes.

- **Cliente**: O cliente envia solicitações ao servidor através de protocolos como HTTP/HTTPS. Ele renderiza a interface do usuário e pode executar lógica de negócios leve ou manipulação de dados antes de exibir as informações ao usuário. Tecnologias

modernas incluem frameworks como React e Angular, que permitem a criação de interfaces dinâmicas e responsivas.

- **Servidor**: O servidor recebe as solicitações do cliente, processa-as e retorna as respostas adequadas. Ele é responsável por tarefas como autenticação, autorização, manipulação de dados e integração com outros serviços. Frameworks como Django e Express.js são comumente usados para construir a camada de servidor.

A web moderna evoluiu significativamente, oferecendo experiências de usuário mais ricas e interativas. Alguns dos principais componentes e práticas incluem:

- **Single Page Applications (SPA)**: SPAs carregam uma única página HTML e dinamicamente atualizam o conteúdo conforme o usuário interage com a aplicação. Isso resulta em uma experiência mais rápida e fluida, semelhante a aplicativos nativos. Frameworks como React e Vue.js são amplamente utilizados para construir SPAs.
- **Progressive Web Apps (PWA)**: PWAs combinam o melhor das experiências web e móvel, permitindo que aplicações web funcionem offline e se comportem como aplicativos nativos. Utilizam tecnologias como Service Workers e Web App Manifests para fornecer essas funcionalidades.
- **APIs RESTful e GraphQL**: As APIs RESTful são padrões para a comunicação entre o cliente e o servidor, permitindo operações de CRUD (Create, Read, Update, Delete) através de endpoints bem definidos. GraphQL oferece uma alternativa, permitindo consultas mais flexíveis e específicas, reduzindo a quantidade de dados transferidos.

Tecnologias de desenvolvimento web:

- **Django**: Django é um framework web Python de alto nível que promove o desenvolvimento rápido e um design limpo e pragmático. Ele oferece uma administração robusta, Object Relational Mapping (ORM) poderoso e diversas ferramentas que facilitam o desenvolvimento de aplicações web seguras e escaláveis (Django, 2023).
- **React**: React é uma biblioteca JavaScript para a construção de interfaces de usuário. Desenvolvida pelo Facebook, ela facilita a criação de componentes reutilizáveis e a gestão eficiente do estado da aplicação, proporcionando uma experiência de usuário dinâmica e responsiva (React, 2023).

- **Celery**: Celery é uma biblioteca Python para execução assíncrona de tarefas e agendamento de trabalhos, permitindo o processamento de tarefas em segundo plano, essencial para operações que não precisam ser imediatas ou que exigem muita computação (Celery, 2023).

2.6.5 Ferramentas de armazenamento de dados

- **Bancos de dados relacionais**: Bancos de dados como PostgreSQL são utilizados para armazenar dados estruturados com integridade referencial e suporte a transações ACID, fundamentais para manter a consistência dos dados em aplicações críticas (PostgreSQL, 2023).
- **Bancos de dados não relacionais**: Bancos de dados NoSQL, como MongoDB, oferecem flexibilidade para armazenar dados semi-estruturados e são ideais para aplicações que exigem escalabilidade horizontal e alta performance em leitura/escrita (MongoDB, 2023).
- **Redis (Caching)**: Redis é um banco de dados de estrutura de dados em memória, usado como cache para melhorar a performance de aplicações web, armazenando resultados de consultas frequentemente acessadas para acesso rápido (Redis, 2023).
- **Data warehouses**: Data warehouses, como Amazon Redshift e Google BigQuery, são utilizados para armazenamento e análise de grandes volumes de dados históricos. Eles permitem a execução de consultas complexas e a geração de insights de negócios a partir dos dados coletados (Amazon Redshift, 2023; Google BigQuery, 2023).

3 MATERIAIS E MÉTODOS

Nesta seção, será apresentado um breve resumo sobre a IRIS e explicaremos os passos que foram tomados para conseguir atingir os objetivos do projeto: planejamento, arquitetura e desenvolvimento. Os resultados dessas etapas serão abordados nas seções 4 e 5.

3.1 A STARTUP IRIS

A IRIS é uma startup que desenvolve soluções de monitoramento da qualidade do ar. O objetivo da empresa é desenvolver um equipamento de monitoramento de baixo custo confiável com tecnologia nacional, que possa ser usado tanto pelo setor público quanto privado.

A startup nasceu a partir de uma pesquisa de doutorado (Campo, 2023), tendo uma base científica sólida para o desenvolvimento do seu produto. A pesquisa buscou criar os fundamentos para o desenvolvimento de uma rede de monitoramento de baixo custo colaborativa, a iniciativa CLEAN: hardware e firmware de aquisição, algoritmo de processamento das medidas e o desenvolvimento de um produto mínimo viável (do inglês Minimum Viable Product, MVP) de uma plataforma para acesso dos dados.

Contudo, o uso de sensores de baixo custo na aplicação de monitoramento da qualidade do ar ainda é recente, e o mercado não está acostumado com esse tipo de solução. Até o momento, a empresa carrega apenas a experiência do MVP executado na tese de doutorado. Por isso, muitas das hipóteses levantadas durante esse MVP ainda precisam ser validadas em aplicações de clientes reais, e para isso é necessário ter um produto que os clientes possam usar e interagir, com muitos requisitos que não são contemplados pela CLEAN.

Dito isso, o momento que a empresa se encontra é o de Product Market Fit, com o objetivo principal de validar seu produto no mercado o quanto antes para poder captar investimento. Desta forma, derivamos a necessidade que este trabalho satisfaz: desenvolver um produto com os requisitos mínimos para conseguir iniciar a operação em um cliente.

Site institucional da empresa: https://iris-ness.com/

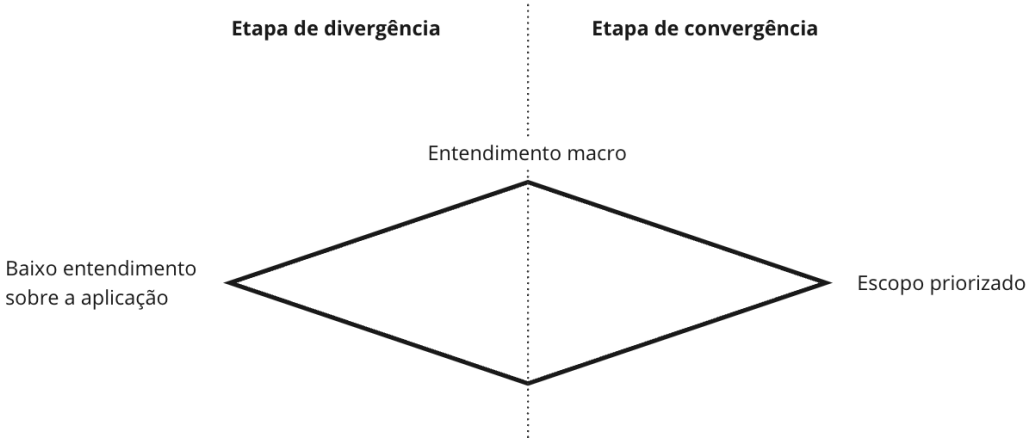
3.2 PLANEJAMENTO E ARQUITETURA DO SISTEMA

No desenvolvimento de um produto ou projeto, a etapa de planejamento deve estar alinhado aos objetivos estratégicos da empresa. No caso do escopo deste trabalho, foram considerados: o momento em que a IRIS se encontra e onde almeja chegar; a maturidade técnica das outras frentes de produtos e serviços da empresa; seus objetivos e metas. Este entendimento holístico afeta diretamente a necessidade por certas funcionalidades, o prazo para a realização de entregas parciais e obtenção de feedbacks do cliente. Com isso em mente, o desenvolvimento deste trabalho foi dividido nas duas etapas que serão elaboradas a seguir: planejamento e definição da arquitetura técnica e implementação do projeto.

O objetivo da etapa de planejamento e arquitetura é levantar todos os requisitos do projeto para que sejam priorizados em escopos bem definidos de futuras versões da plataforma e do equipamento. O entendimento das regras de negócio, dos requisitos básicos, e de como o produto pode escalar e crescer no futuro é essencial para construir uma arquitetura eficiente que suporte este crescimento, ao mesmo tempo que permita entregas parciais para a obtenção de feedback rápido.

O planejamento foi executado em duas fases. Na primeira, o objetivo foi aumentar o entendimento do autor sobre a aplicação e explorar os possíveis caminhos que o projeto poderia seguir; na segunda fase, o foco foi convergir os resultados obtidos na primeira fase em uma arquitetura técnica e um escopo priorizado. Para isso foram seguidos 9 passos que serão documentados neste projeto, cada um em um item da seção “4. Resultados: Arquitetura e Planejamento do Projeto”.

Figura 15 - Fluxograma da metodologia de planejamento de projeto usada



Fonte: Elaborado pelo autor (2024)

Partimos de um baixo entendimento da aplicação, e seguimos os seguintes passos para alcançar um bom entendimento macro na etapa de divergência:

- Entendimento da proposta de valor em torno da problemática chave que a IRIS resolve (abordado na seção 4.1.1);
- Segmentação das necessidades por perfil de cliente (abordado na seção 4.1.2);
- Benchmarking com plataformas similares (abordado na seção 4.1.3);
- Listagem dos requisitos funcionais (abordado na seção 4.1.4);
- Desenho dos wireframes (abordado na seção 4.1.5).

Com a visão macro definida, conseguimos convergir em um escopo priorizado e planejamento do projeto. Para um bom planejamento, é necessário ter tomado a maior parte das decisões técnicas e de arquitetura, uma vez que estas decisões impactam em como os requisitos serão implementados. Assim, os resultados esperados ao fim da etapa de convergência são:

- Blueprint da arquitetura;
- Definição das interfaces de comunicação entre Firmware / Backend / Frontend e das regras de negócio envolvidas.
- Breakdown da arquitetura em tarefas definidas em: resultado final esperado da tarefa; como esta deve ser implementa; estimativa de esforço;
- Divisão do trabalho em entregas parciais para a área comercial.

Para chegar nesses resultados foram executados os seguintes passos:

- Modelagem dos domínios da aplicação por meio de técnicas de Domain Driven Design (abordado na seção 4.2.1);
- Definição das interfaces de interação entre as entidades e principais regras de negócio (abordado na seção 4.2.2);
- Escolha das ferramentas e desenho da arquitetura ideal (abordado na seção 4.2.3);
- Breakdown em tarefas e estimativa temporal seguindo princípios da metodologia ágil e priorização final do escopo (abordado na seção 4.2.4);

3.3 DESENVOLVIMENTO DO SISTEMA

Com as principais decisões técnicas tomadas na etapa de arquitetura do sistema, resta ao desenvolvimento tomar decisões pontuais sobre a melhor forma de implementar as regras de negócio e interfaces definidas. Sendo assim, neste documento, serão descritos apenas os principais desafios implementados.

Um ponto a ser ressaltado foi o uso de princípios de Test Driven Development na implementação das regras de negócio mais cruciais da aplicação: segurança/acesso aos dados; consistência das informações armazenadas; e processamento e calibração dos dados.

4 RESULTADOS: PLANEJAMENTO E ARQUITETURA DO SISTEMA

Como descrito nos materiais e métodos, o trabalho desta seção se divide em duas grandes etapas, cujos resultados são apresentados nesta seção:

- **Etapa de divergência:** entendimento sobre o negócio, lista de requisitos funcionais que fazem parte do projeto e um wireframe das telas do sistema.
- **Etapa de convergência:** arquitetura ideal do sistema, definição das interfaces e regras de negócio, quebra da arquitetura em unidades de trabalho mínimas (tasks) com plano de implementação definido e tempo estimado.

4.1 ETAPA DE DIVERGÊNCIA: LEVANTAMENTO DOS REQUISITOS

A fim de manter o foco nos trabalhos técnicos realizados, concentrados principalmente na etapa de convergência, o trabalho por trás dessa etapa será explicado de forma breve.

4.1.1 Proposta de valor em torno da problemática chave

Foi realizado um brainstorming com os sócios da empresa (APÊNDICE A) para entender a como a IRIS resolve as dificuldades em torno da problemática chave: Dificuldade em criar e operar redes de monitoramento atmosférico;

O brainstorming foi estruturado quebrando a problemática chave em dificultadores, e para cada um desses dificultadores, as soluções que a IRIS pode trazer. Os requisitos finais devem estar em linha com as dificuldades levantadas. Por exemplo, uma visualização da calibração do equipamento, e comparação com dados de estações de referência ajudaria na resolução da dificuldade do item 4 abaixo.

Principais dificuldades dos stakeholders levantadas nesse processo:

1. Padronizar e consolidar de dados das diferentes fontes de medição;
2. Compartilhar e aumentar a transparência sobre dados de emissões;
3. Acessar em tempo real das informações;
4. Confiar nas informações geradas pelos equipamentos de baixo custo;
5. Gerar insights relevantes e planos de ação sobre os dados;
6. Gerir grandes volumes de dados;
7. Garantir a segurança dos dados gerados;

4.1.2 Segmentação das necessidades por stakeholder

De maneira similar, foi feita uma quebra das necessidades e dificuldades por stakeholder: Órgãos Públicos, Indústrias, Universidades e Sociedade Civil (APÊNDICE B).

Apesar de que a aplicação do uso dos dados e alguns requisitos de funcionalidade variam entre stakeholders, notamos que a necessidade central que o sistema deve atender é compartilhada: orquestrar a aquisição de dados dos equipamentos, servir como sistema central de acesso aos dados e ser capaz de transformar dados em informações e insights.

4.1.3 Benchmarking com Plataformas Similares

Esse estudo foi documentado na seção de Fundamentação Teórica. Do estudo, foram derivados insights de possíveis funcionalidades e modelagem da tecnologia.

Funcionalidades compartilhadas em comum por todas as plataformas, que nossa plataforma deve replicar:

1. Compilação de índices da qualidade do ar;
2. Dashboard com visão espacial das estações, detalhada de uma estação e mecanismo de filtragem;
3. Capacidade de consolidação de dados de diferentes fontes.

Melhores práticas de modelagem da tecnologia identificadas:

1. Modelagem de pipeline de ETL fléxivel da OpenAQ;
2. Modelagem de tecnologia de armazenamento de dados da OpenAQ (Timescale + PostGIS + S3);
3. Conectividade e resiliência energética dos dispositivos da Kunak;
4. Capacidade modular de slots de sensores da Kunak.

Em relação ao CLEAN, algumas melhorias que nosso sistema pode propor:

1. Delegar as de responsabilidades do firmware ao backend:
    a. Gerenciamento dos datasheets dos equipamentos. Facilitando a manutenção e gerenciamento da frota.
    b. Processamento dos dados, restando ao firmware o envio dos dados brutos. A aplicação do Smoother (Campo, 2023) ainda deve ser do firmware dado que atua como um filtro passa-baixas e serve como uma suavização do ruído analógico.

2. Construção de um sistema escalável: implementação de tecnologias de processamento e armazenamento de dados focadas em grandes volumes de dados em estrutura de séries temporais.
3. Implementação de um pipeline de calibração dos dados: graças ao trabalho exploratório da iniciativa CLEAN, foi possível construir uma metodologia de calibração da IRIS que pode agora ser implementada em um sistema.
4. Implementação de requisitos não priorizados para o Clean, pelo caráter não comercial da aplicação, como:
    a. Conceitos de gerenciamento de equipes e usuários;
    b. Segurança na camada de aquisição dos dados;
    c. Segurança na camada de aplicação (cloud);
    d. Regras de row level security nos dados;

4.1.4 Listagem Dos Requisitos Funcionais E Não Funcionais

Os requisitos listados abaixo descrevem funcionalidades que o sistema da IRIS deve possuir para ser um produto estabelecido no mercado. Esta é uma lista viva e que deve ser constantemente atualizada de acordo com as interações com parceiros e clientes.

Requisitos funcionais:

Para o escopo deste projeto, foram separadas as funcionalidades em "*Must have" / "Nice to have"*. A priorização levará em conta não apenas esse aspecto funcional, mas também aspectos de implementação e modelagem do software.

Tabela 1 - Listagem dos requisitos funcionais (sem priorização)

Categoria / Requisito		Importância
Conectividade e Coleta dos dados		
	O equipamento deve ser capaz de enviar dados em tempo real ao servidor	Must have
	O sistema deve receber dados de sensores com períodos de amostragem diferentes	Must have
	O sistema deve ter um mecanismo de redundância para envio de dados em caso de falha na conexão.	Nice to have
Acessos e Gerenciamento de Usuários		
	O sistema deve permitir o registro e gerenciamento de clientes	Must have
	Cada cliente deve ter acesso a um ambiente exclusivo	Must have
	Usuários podem acessar o ambiente de mais de um cliente	Nice to have
	Cada usuário deve ter um papel dentro do time (e.g.: admin/analista/operador/externo/...) com diferentes permissões	Nice to have

Categoria / Requisito		Importância
Gerenciamento da Frota		
	É necessário manter um registro dos equipamentos montados	Nice to have
	É necessário manter um registro dos sensores adquiridos e em quais equipamentos os sensores foram instalados	Nice to have
	Um equipamento pode ser alugado ou vendido para um cliente. Isso será chamado de alocação	Nice to have
	O sistema deve gerar alertas quando um sensor ou equipamento necessita de manutenção.	Nice to have
	O sistema deve permitir a fácil atualização de firmware dos sensores remotamente.	Nice to have
	O sistema deve ser capaz de operar equipamentos de terceiros, por exemplo, estações de referência	Nice to have
Flexibilidade do equipamento		
	O equipamento comporta diferentes configurações de sensores	Must have
Processamento das medidas		
	Cada sensor tem constantes definidas em datasheet que serão utilizadas para o processamento dos dados	Must have
	Os dados brutos gerados por cada sensor devem ser processados de acordo com as recomendações do fabricante para o sensor	Must have
	O sistema deve manter rastreabilidade dos dados processados	Must have
	O sistema deve permitir a aplicação de algoritmos de calibração para melhorar a precisão dos dados dos sensores.	Must have
Integração de bases de dados		
	O sistema deve ser capaz de se conectar a sistemas de monitoramento já existentes (públicos ou privados)	Must have
	O sistema deve poder consumir/consolidar dados de outras plataformas	Must have
	O sistema deve poder fornecer dados para outras plataformas	Must have
	O sistema deve permitir a exportação dos dados em diferentes formatos (e.g., CSV, JSON, XML) para análise externa.	Must have
Interoperabilidade		
	O sistema deve ser compatível com normas e padrões nacionais (CONAMA) de monitoramento ambiental.	Must have
Notificações		
	Permitir a configuração dinâmica de limites em uma granularidade temporal para cada variável do sistema	Nice to have
	O sistema deve permitir a configuração de diferentes níveis de severidade para as notificações.	Nice to have
Dashboard/Análise dos dados		
	Analisar dados de estações de baixo custo e estações de referência	Must have
	Visualização espacial das estações resumidas	Must have
	Visualização detalhada das estações	Must have

Categoria / Requisito		Importância
	O sistema deve mostrar as concentrações ao longo do tempo do tempo	Must have
	Capacidade de comparar as medições entre estações de referência e baixo custo co-locadas para fins de validação da calibração	Nice to have
Visibilidade dos Dados		
	Somente o cliente tem acesso aos dados gerados pelo equipamento no período em que o equipamento estava alocado, exceto que o cliente decida publicas esses dados.	Must have
	Um cliente pode habilitar uma página pública na qual qualquer usuário, mesmo não autenticado teria acesso (por exemplo, um orgão público disponibilizando os dados para a sociedade civil)	Nice to have
	Os dados disponíveis na página pública do cliente serão habilitados por alocação de equipamento.	Nice to have
Gerenciamento das Calibrações		
	A calibração de um equipamento é atrelada aos sensores instalados no equipamento;	Must have
	Um equipamento pode ser recalibrado;	Nice to have
	Orquestrar o treinamento de modelos de calibração na nuvem	Nice to have
Integração de diferentes tipos de fontes		
	Satélite	Nice to have
	Inventário de emissões	Nice to have
	Modelagens	Nice to have
Monitoramento de ambientes Internos		Nice to have

Fonte: Elaborado pelo autor (2024)

Requisitos não funcionais:

Tabela 2 - Listagem dos requisitos não funcionais

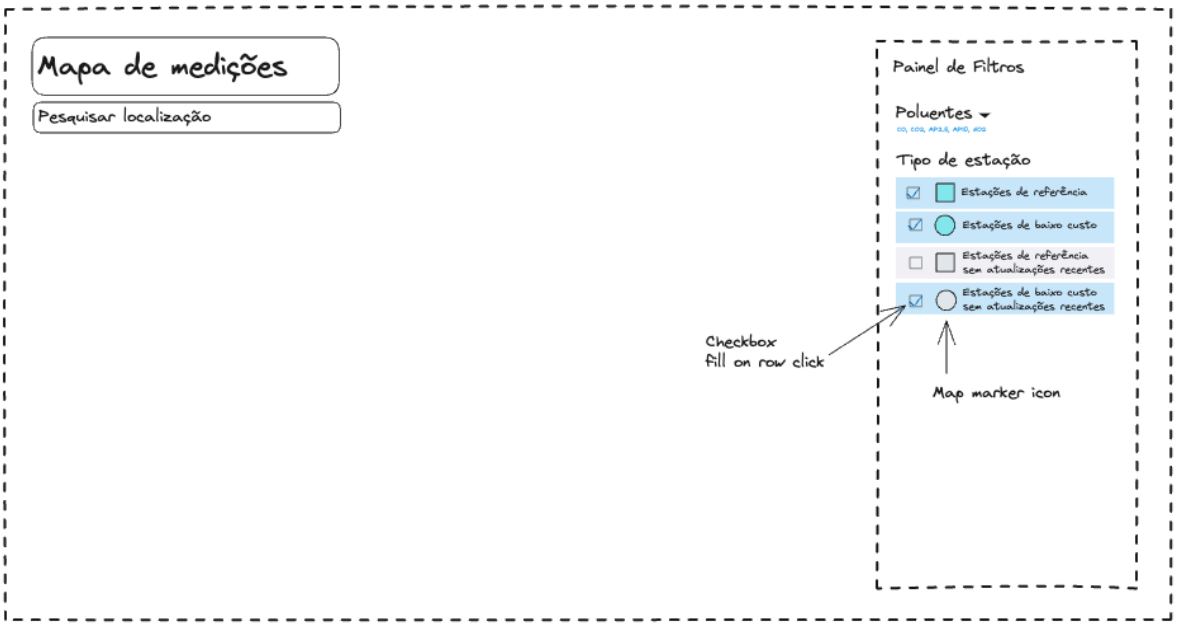
Categoria	Requisito
Performance	O sistema deve garantir tempos de resposta rápidos.
Escalabilidade	O sistema deve escalar para altos volumes de ingestão, armazenamento e processamento dos dados.
Disponibilidade	O sistema deve ter uma arquitetura de alta disponibilidade (HA) para garantir a continuidade do serviço.
Flexibilidade	O sistema deve permitir fácil adição de novos tipos de sensores e funcionalidades sem necessidade de grandes reconfigurações.
Resiliência	O sistema deve ser capaz de se recuperar automaticamente de falhas de componentes sem intervenção manual.

Fonte: Elaborado pelo autor (2024)

4.1.5 Desenho dos wireframes

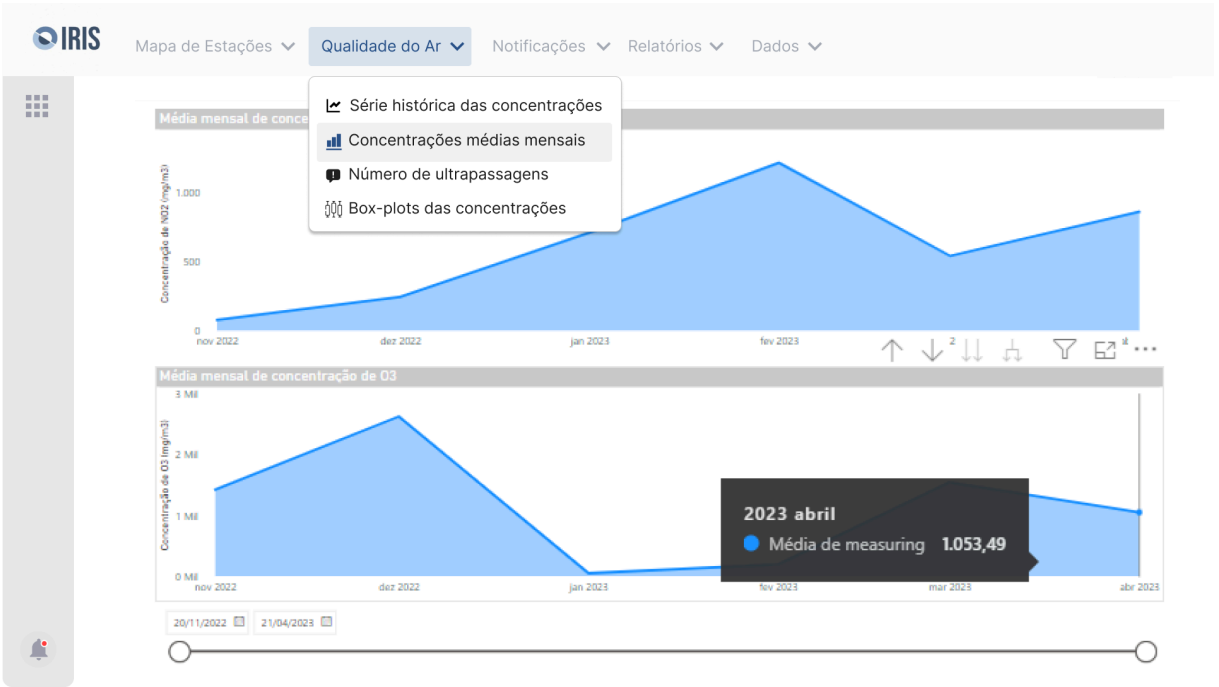
Com os requisitos definidos, podemos criar esboços das telas finais da aplicação (APÊNDICE I). A figura 16 e 17 apresentam um exemplo de uma das telas esboçadas.

Figura 16 - Esboço da visão de mapa na plataforma



Fonte: Elaborado pelo autor (2024)

Figura 17 - Wireframe da visão detalhada de uma estação na plataforma



Fonte: Elaborado pelo autor (2024)

4.2 ETAPA DE CONVERGÊNCIA: ARQUITETURA DO SISTEMA E DEFINIÇÃO DO ESCOPO

Arquitetar um sistema escalável, tanto do ponto de vista de carga, como do ponto de vista funcional é um assunto delicado. Pois as características que o tornam escalável estão geralmente profundamente enraizadas em seu mecanismo fundamental. Todos os outros requisitos são afetados pelas decisões de arquitetura tomadas aqui, não apenas em termos de desempenho, mas também na forma como serão implementados. Em outras palavras, é essencial acertar essa parte, caso contrário, pode-se acabar com uma ferramenta inútil uma vez que a demanda por dados cresça, ou a demanda por novas funcionalidades conflite com a modelagem inicial.

Uma boa arquitetura também habilita uma etapa de desenvolvimento suave, com tarefas específicas e bem definidas tanto em relação aos requisitos funcionais como também os não funcionais. Além disso, garante para a área de negócios visibilidade sobre o cronograma, o que impacta diretamente as ações comerciais.

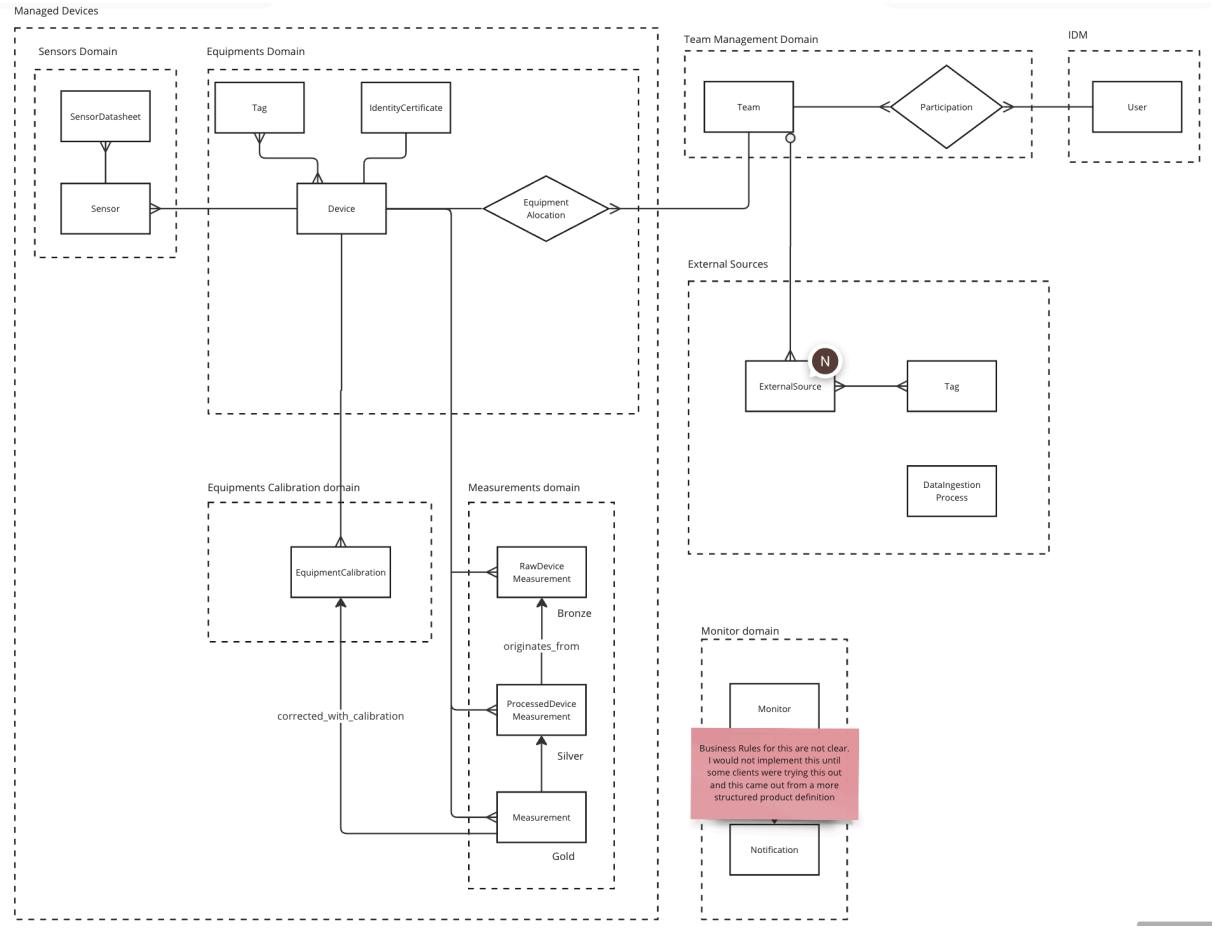
Um ponto importante a ser ressaltado é que apesar dos diferentes caminhos que a aplicação pode seguir no futuro (e.g.: operar estações de referência) um grande problema que gera atrasos em projetos de engenharia é a aplicação de otimizações prematuras. No contexto da IRIS, o maior custo que a empresa tem é o custo de oportunidade da inoperância. Qualquer retrabalho futuro que tenha gerado um leve desperdício de tempo no presente tem um custo em uma ordem de grandeza muito menor que o custo de oportunidade do projeto.

Esta é uma das premissas que guiam o desenvolvimento da arquitetura e priorização do escopo: como montar um projeto que seja resiliente à evoluções e mudanças na regra de negócio, ao mesmo tempo que possui entregas rápidas para habilitar a área comercial o quanto antes.

4.2.1 Modelagem Dos Domínios Da Aplicação

Usando técnicas de Domain Driven Design a aplicação foi modelada nos seguintes domínios.

Figura 18 - Desenho dos domínios da aplicação



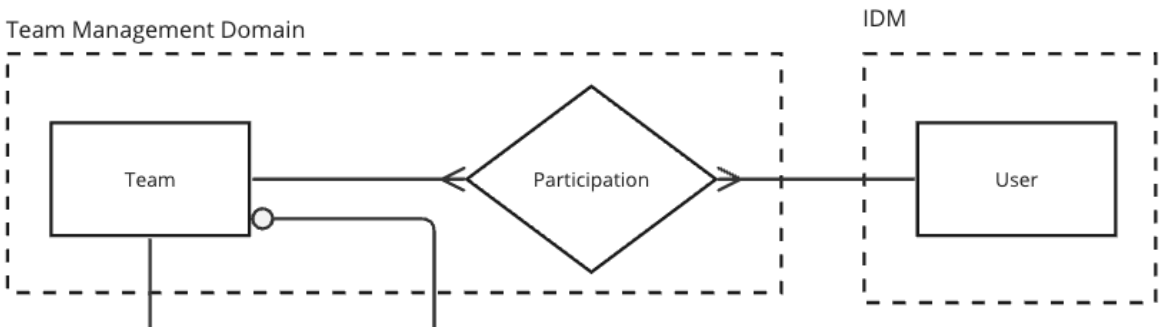
Fonte: Elaborado pelo autor (2024)

Esses domínios podem, posteriormente, ser implementados por microsserviços correspondentes. Analisando cada um desses domínios com mais profundidade:

4.2.1.1 Domínios de gestão de clientes e identity management

Esses domínios seguem a modelagem padrão do mercado para os requisitos. Um User pode ter a atuação em diferentes Teams, representada pela entidade Participation, assumindo Roles nessa participação. Um exemplo seria um consultor externo que opera equipamentos em N clientes, ou um usuário convidado que a empresa possa convidar para ter acesso a um subconjunto dos dados.

Figura 19 - Desenho do domínio de Identity Management e Gerenciamento de Equipes

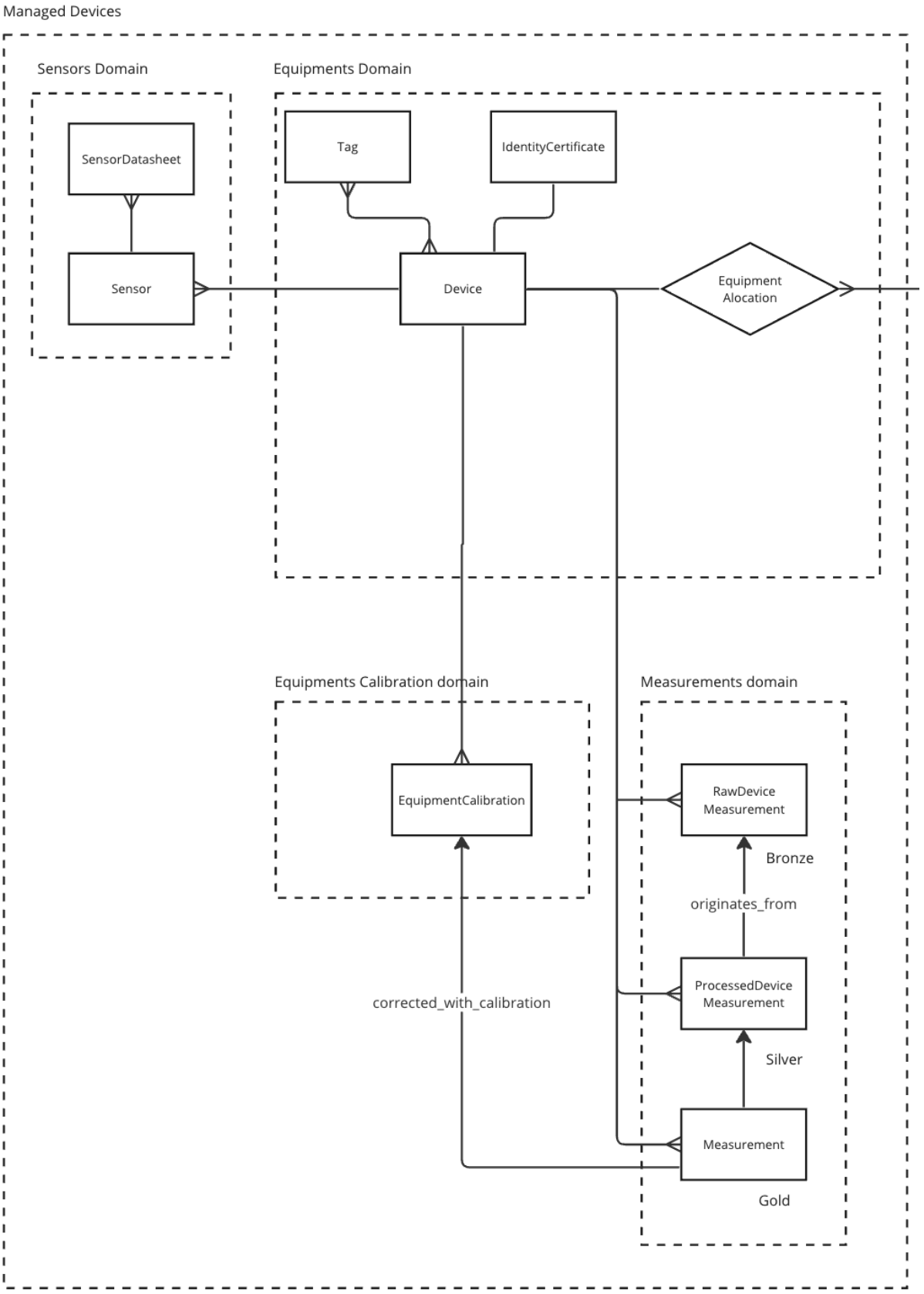


Fonte: Elaborado pelo autor (2024)

4.2.1.2 Domínios dos equipamentos gerenciados pela IRIS

Esse domínio é onde maior parte do escopo da aplicação se encontra e pode ser quebrado em 4 subdomínios:

Figura 20 - Desenho do domínio de equipamentos gerenciados



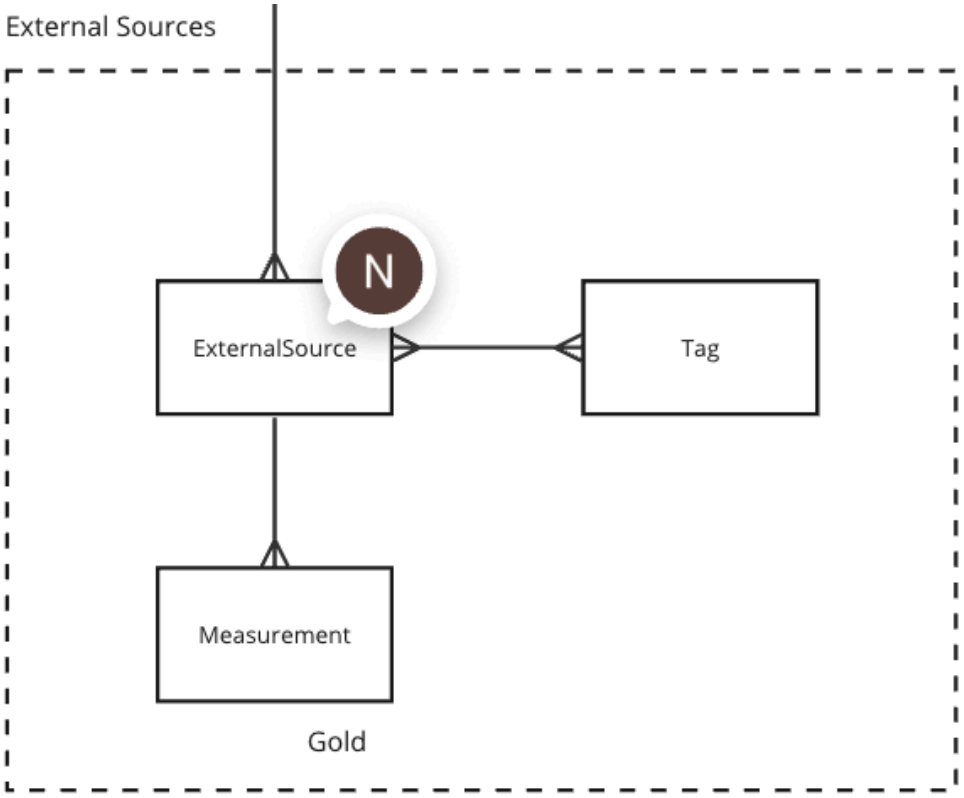
Fonte: Elaborado pelo autor (2024)

- **Sensors Domain**
    - **SensorDatasheet**: Contém as especificações técnicas do sensor, como características e parâmetros de funcionamento.
    - **Sensor**: Representa o sensor físico que coleta dados ambientais. Ele está vinculado a um SensorDatasheet que descreve suas especificações.
- **Equipments Domain**
    - **Device**: Representa o dispositivo de medição.
    - **DeviceAllocation**: Representa a relação de alocação de um equipamento em um cliente por um período específico de tempo.
    - **Tag**: Metadados que podem ser usados para gerenciar grupos de dispositivos. Por exemplo, agrupamento por órgãos ambientais ou redes conectadas de equipamentos.
    - **IdentityCertificate**: Certificado que autentica a identidade do dispositivo, garantindo que ele é genuíno e autorizado a operar na rede.
- **Equipments Calibration Domain**
    - **EquipmentCalibration**: Cada equipamento tem seu arquivo de calibração gerado a partir da comparação com uma estação de referência. Essa calibração deve ser aplicada nas medidas compensadas a fim de chegar ao valor final de concentração.
- **Measurements Domain:** Esse domínio modela os dados seguindo a arquitetura Medallion. Desta forma é possível ter o traceback das transformações, ao mesmo tempo que podem ser corrigidas falhas no processo de maneira retroativa.
    - **RawDeviceMeasurement:** Dados brutos coletados diretamente do dispositivo sem qualquer processamento ou correção.
    - **ProcessedDeviceMeasurement**: Dados que foram processados e corrigidos com base nos algoritmos de compensação do equipamento determinada em laboratório.
    - **Measurement**: Dados finais e refinados que estão prontos para análise e tomada de decisão. Estes dados passaram por todos os estágios de processamento e foram corrigidos pelo algoritmo de calibração associado ao equipamento.

4.2.1.3 Domínio de gerenciamento de fontes externas de dados

Este domínio tem uma estrutura similar ao domínio de equipamentos gerenciados, porém mais simples. No entanto, o objetivo, como destacado na etapa de divergência, é ter apenas uma prova de conceito a fim de tangibilizar em reuniões comerciais a capacidade de expansão do sistema. É previsto que ao vender o primeiro projeto, este domínio precise ser retrabalhado ou refinado dado que ele tem potencial de ser mais complexo.

Figura 21 - Desenho do domínio de fontes externas



Fonte: Elaborado pelo autor (2024)

Ao desenhar as versões iniciais desses domínios, algumas perguntas não identificadas nas etapas anteriores surgiram. Por exemplo: Como lidar com as re-calibrações dos equipamentos? Como lidar com a publicidade de dados e regras de acesso aos dados dos equipamentos dentro de um cliente?

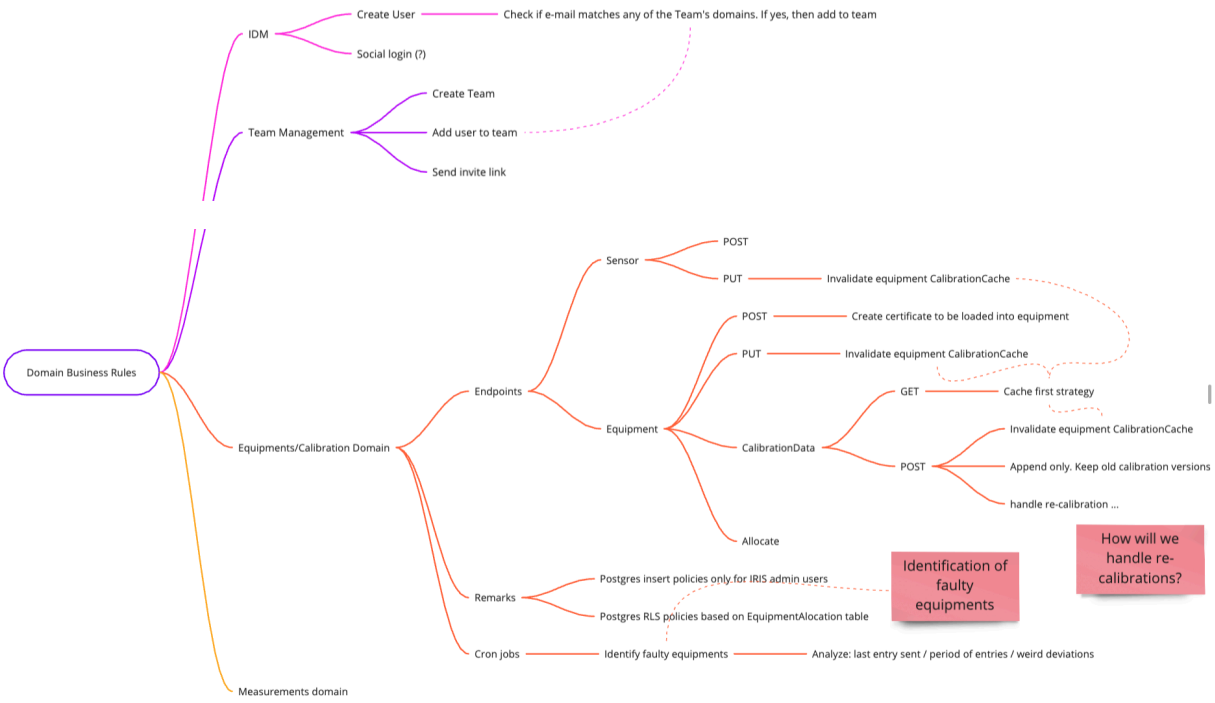
Esse processo iterativo com os sócios (experts do domínio) fazem com que a modelagem alcance uma versão final robusta.

4.2.2 Definição Das Interfaces e Regras De Negócio

O objetivo deste passo é entender as interfaces de comunicação entre as entidades da aplicação e desenhar as regras de negócio por trás dessas interações. Essas interfaces se darão na maior parte através de APIs REST.

Na figura 22, o detalhamento dessas regras de negócio pode ser encontrado. Nota-se que começam a surgir uma série de requisitos não funcionais como o uso de uma estratégia de cache a fim de reduzir a latência, e consequentemente seus mecanismos de invalidação.

Figura 22 - Esboço das interfaces e regras de negócio por domínio

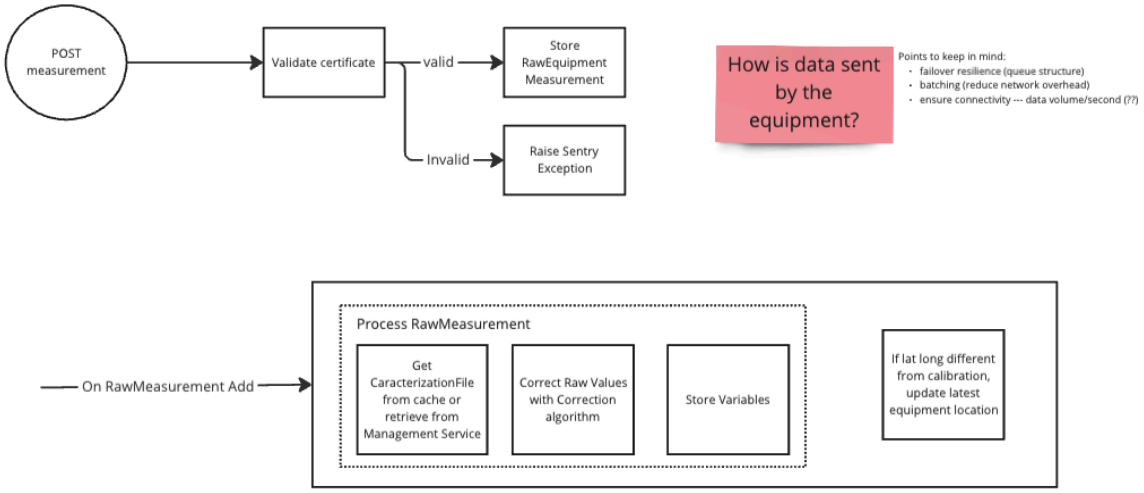


Fonte: Elaborado pelo autor (2024)

Quando se fala de monitoramento da qualidade do ar de baixo custo, a principal regra de negócio da aplicação é em relação ao processamento dos dados dos sensores eletroquímicos. De acordo com Campo (2023), a transformação das medições de tensão em valores de concentração segue dois processos: correção a partir dos algoritmos de compensação do fabricante; calibração por modelo de IA, treinado a partir de comparação com dados gerados por estações de referência. O processo de coleta de uma medida é ilustrado na figura abaixo. Nele, podemos entender a necessidade pela padronização de um processo de transformação multi-etapas, e a criação de interfaces de processamento dos dados

brutos de cada sensor, dos dados consolidados determinados pelas equações de caracterização multi-variáveis fornecidas pelo fabricante, e uma transformação final a partir do modelo de IA treinado para o equipamento.

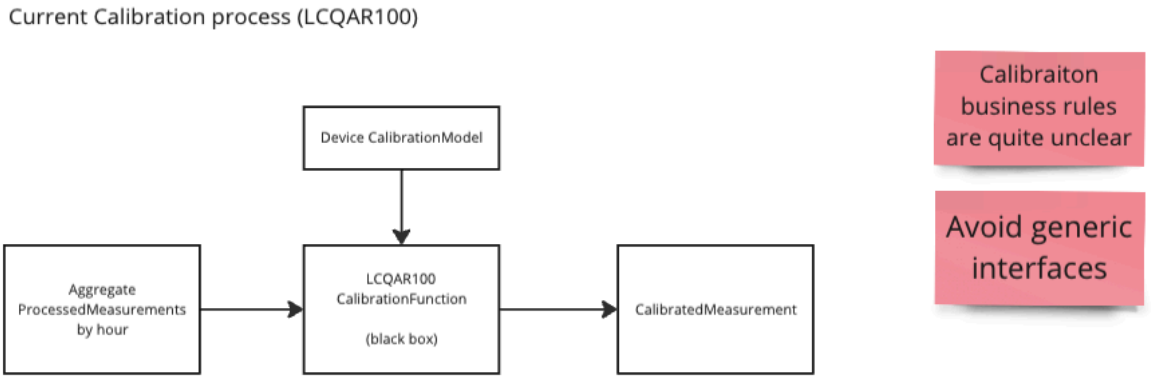
Figura 22 - Fluxograma das regras de negócio da aquisição e processamento das medidas



Fonte: Elaborado pelo autor (2024)

O processamento dos dados brutos de um sensor é um processo bem definido e estabelecido. No entanto, a calibração é uma frente técnica com baixa maturidade técnica na empresa ainda. Desta forma, no desenvolvimento é importante não implementar otimizações prematuras e fazer uma solução que não seja genérica.

Figura 23 - Fluxograma simplificado do processo de calibração

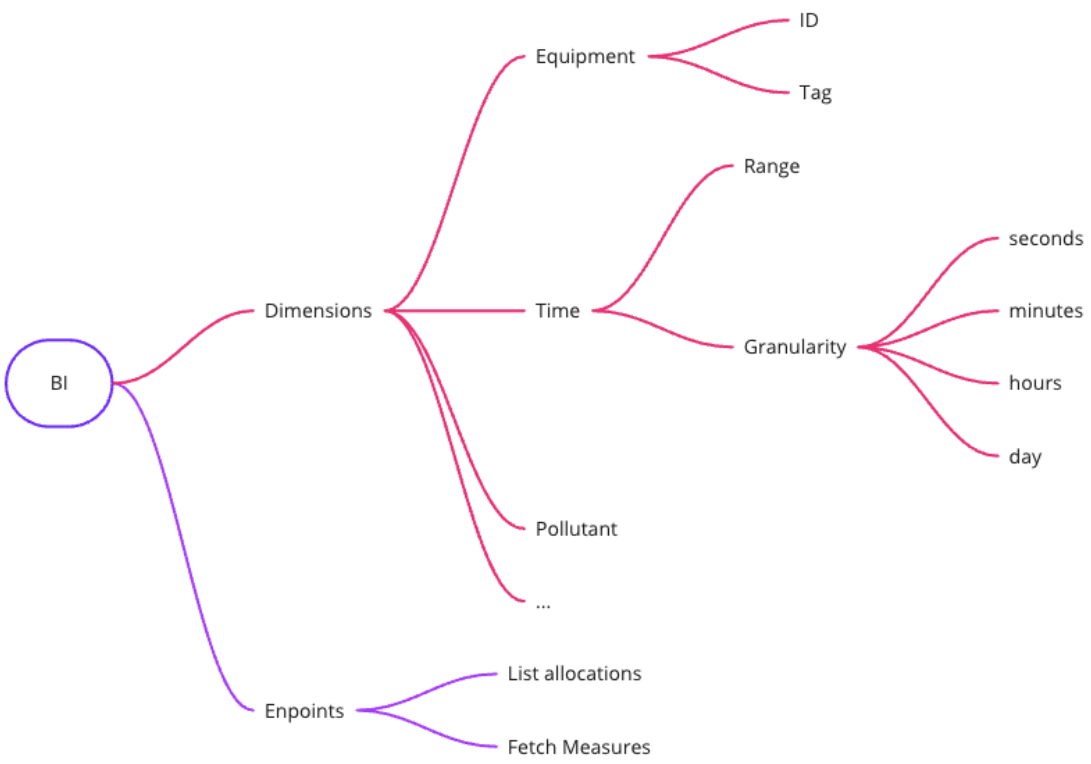


Fonte: Elaborado pelo autor (2024)

As regras de negócio acima envolvem principalmente a frente de gerenciamento dos equipamentos e armazenamento de medidas. No entanto, é importante ressaltar que outro aspecto importante de ser otimizado em um projeto de IoT, além da ingestão desses dados, é a análise destes. Pensando nisso, foi pensado também nas formas como esses dados serão consumidos, impactando diretamente na modelagem da tecnologia que será usada.

Fazendo um paralelo de um modelo estrela do sistema proposto, onde o fato é a medida, podemos pensar nas seguintes dimensões e informações que serão consumidas:

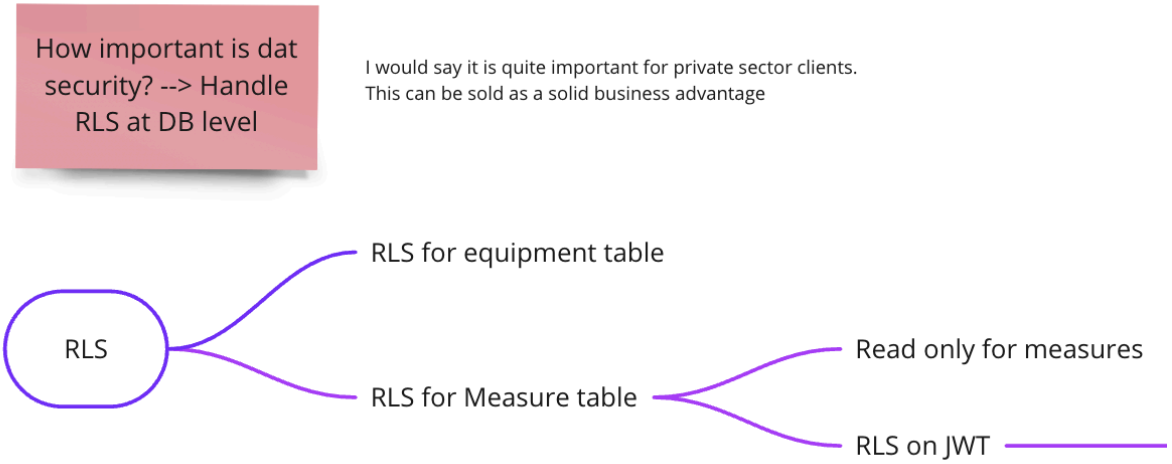
Figura 24 - Esboço das dimensões do modelo estrela e endpoints de consulta chamados pelas aplicações de BI



Fonte: Elaborado pelo autor (2024)

A figura 25 esboça algumas soluções de Row-Level Security:

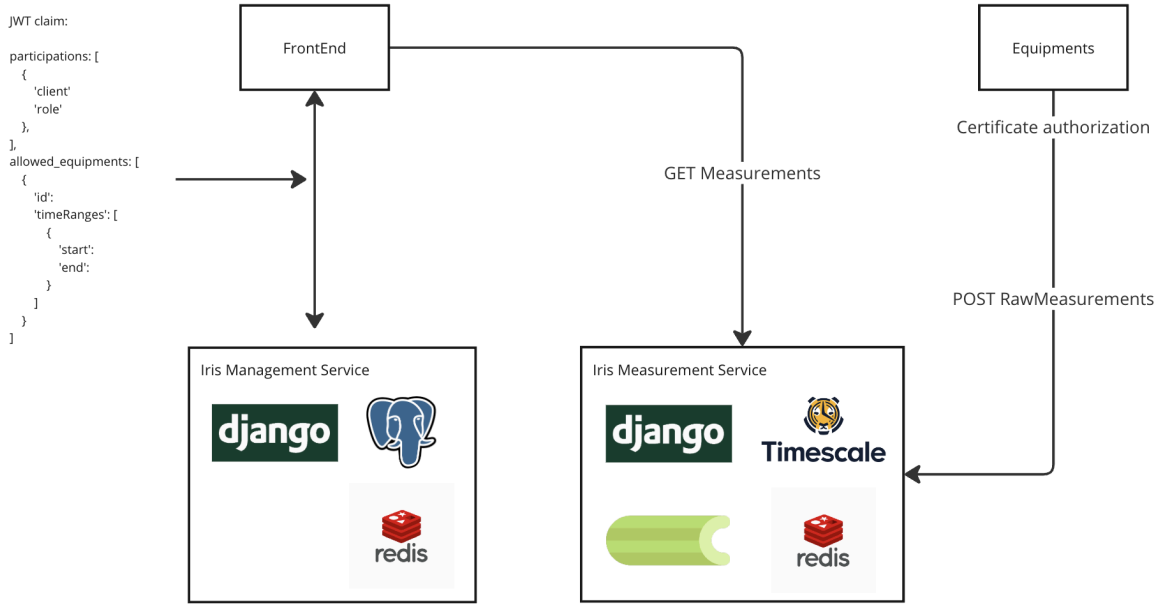
Figura 25 - Esboço das possíveis regras de Row Level Security (RLS)



Fonte: Elaborado pelo autor (2024)

Essas definições levantam aspectos importantes que devem ser considerados no desenho da arquitetura final. Imagine o cenário onde cada domínio é implementado seguindo o pattern de microservice-per-domain e o pattern de database-per-microservice. Nesse caso hipotético, não enfrentamos dificuldade nas regras de negócio de gerenciamento e coleta dos dados dos equipamentos, mas o consumo dessas informações se tornaria muito mais complexo, uma vez que estaríamos consolidando dados de múltiplos serviços em nosso dashboard. O consumo das informações demandaria uma camada de Data Warehouse ou Data Federation na qual consultas que executam joins em tabelas de múltiplos serviços poderiam ser executadas. Uma das ideias iniciais quando se iniciou o desenvolvimento do projeto era a implementação do backend em dois serviços:

Figura 26 - Esboço de uma arquitetura de microserviços, na qual a ingestão das medidas é implementada por um serviço diferente da gestão dos equipamentos.



Fonte: Elaborado pelo autor (2024)

Uma das principais vantagens dessa implementação é que o serviço de medidas tem necessidades de tecnologia e escala diferentes. Por exemplo, poderia ser usado um DB otimizado para dados temporais como o InfluxDB, enquanto um BD relacional atende melhor aos requisitos do serviço de gerenciamento. No entanto, essa divisão traria uma série de complexidades na análise dos dados como comentado anteriormente.

4.2.3 Comunicação Com O Equipamento

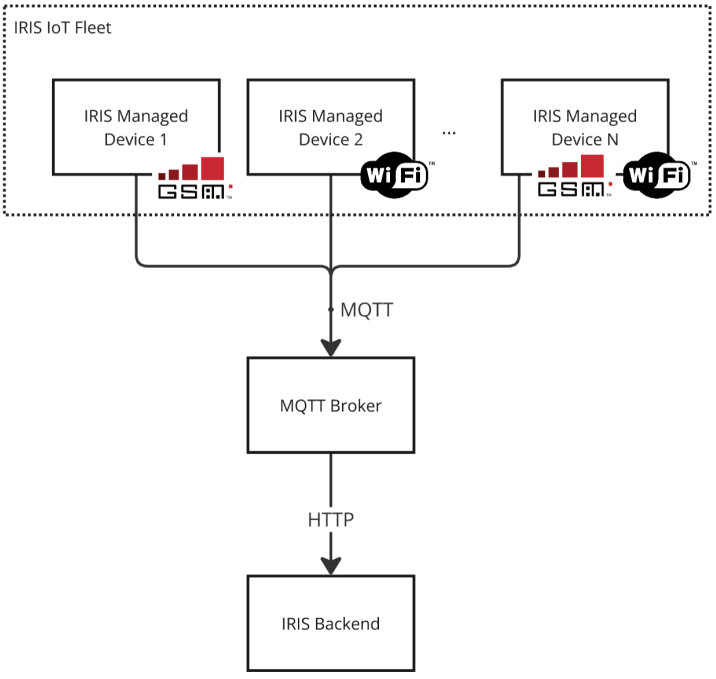
A eletrônica do equipamento foi contratada através de uma empresa terceirizada. Portanto, a responsabilidade deste trabalho era definir o escopo da integração, gerenciar o desenvolvimento e garantir que fosse integrado à plataforma.

- **Conectividade**: O equipamento já possui módulo de conexão GPRS e Wi-Fi. Devido a necessidade da calibração dos equipamentos, qualquer equipamento vendido hoje, não seria instalado em um cliente em pelo menos 4 meses, entre o tempo de montagem do hardware e calibração do sistema. Durante a calibração, o equipamento teria acesso a Wi-Fi e GPRS uma vez que estaria instalado ao lado de uma estação de referência.

Foi estudado de forma breve o trabalho para o desenvolvimento de um módulo LoRaWAN e de Internet Satelital. A conclusão foi que seria possível adicionar esses módulos a tempo e essa decisão seria melhor tomada no momento da venda.

- **Protocolo de Comunicação:** Foi estudado o uso de MQTT e HTTPs. Apesar de o MQTT ser mais adequado para o cumprimento dos requisitos da aplicação, demandaria um esforço maior para a implementação, com um ganho pequeno. Como esta não é uma decisão estrutural, não seria um grande retrabalho mudar para uma arquitetura MQTT no futuro. Isso poderia inclusive ser feito sem alterações no servidor, no qual o MQTT broker poderia fazer a tradução das requisições para HTTP permitindo uma transição gradual.

Figura 27 - Esboço de arquitetura para adaptar comunicação dos equipamentos com o servidor de HTTP para MQTT sem refatorar o servidor



Fonte: Elaborado pelo autor (2024)

- **Interface de Comunicação**

Foram definidas duas versões do payload enviado pelo equipamento. As duas interfaces conseguem passar a mesma informação, diferindo apenas na estrutura. Uma mais legível para um humano (v1), outra otimizando o tamanho do pacote (v2).

Essas interfaces permitem que um equipamento tenha N sensores, e flexível para qualquer modelo de sensor.

Figura 28 - Interface JSON definida para o payload do equipamento, com foco em legibilidade, implementada pela API V1

Payload v1

```
// V1
// HEADER
// content-type: text/json

// BODY
_ = {
    "device_mac_address": "",
    "measurements": [
        {
            "lat": "",
            "long": "",
            "timestamp": "",
            "sensor_readings": [
                {
                    "sensor_serial_number": "",
                    "reading_data": {
                        "temperature": ""
                    }
                },
                {
                    "sensor_serial_number": "",
                    "reading_data": {
                        "aeValue": "",
                        "weValue": ""
                    }
                },
                {
                    "sensor_serial_number": "",
                    "reading_data": {
                        "temperature": "",
                        "humidity": "",
                        "histogram": [
                            0,
                            "...",
                            11
                        ]
                    }
                },
            ]
        }
    ]
};
```

Fonte: Elaborado pelo autor (2024)

Figura 29 - Interface CSV definida para o payload do equipamento, com foco em redução do pacote de dados, implementada pela API V2

```
1   V2
2
3   HEADER
4   content-type: text/csv
5   device-mac-address:
6
7   BODY
8   ts,lat,lng,0,1,2,3,4,5,6,7,8,9
9   ,,,id1,id2,id3,,,,,
10  t0,,,20,20;30,10;20;30;40;50,,,,,
11  t1,,,19,22;28,,,,,,
12  t2,,,20,22;28,,,,,,
13  t3,,,19,20;30,30,10;20;30;40;50,,,,,
```

Fonte: Elaborado pelo autor (2024)

No V2, cada coluna representa um dos sensores. Caso o dado gerado pelo sensor seja vetorial e não um valor escalar, este é dividido pelo caractere “;”.

- **Resiliência:** A implementação do firmware foi realizada pela consultoria externa. Portanto, foi de responsabilidade deste trabalho, sabatinar a empresa sobre os algoritmos de sincronização das medidas após quedas de energia / sinal e testar as entregas.

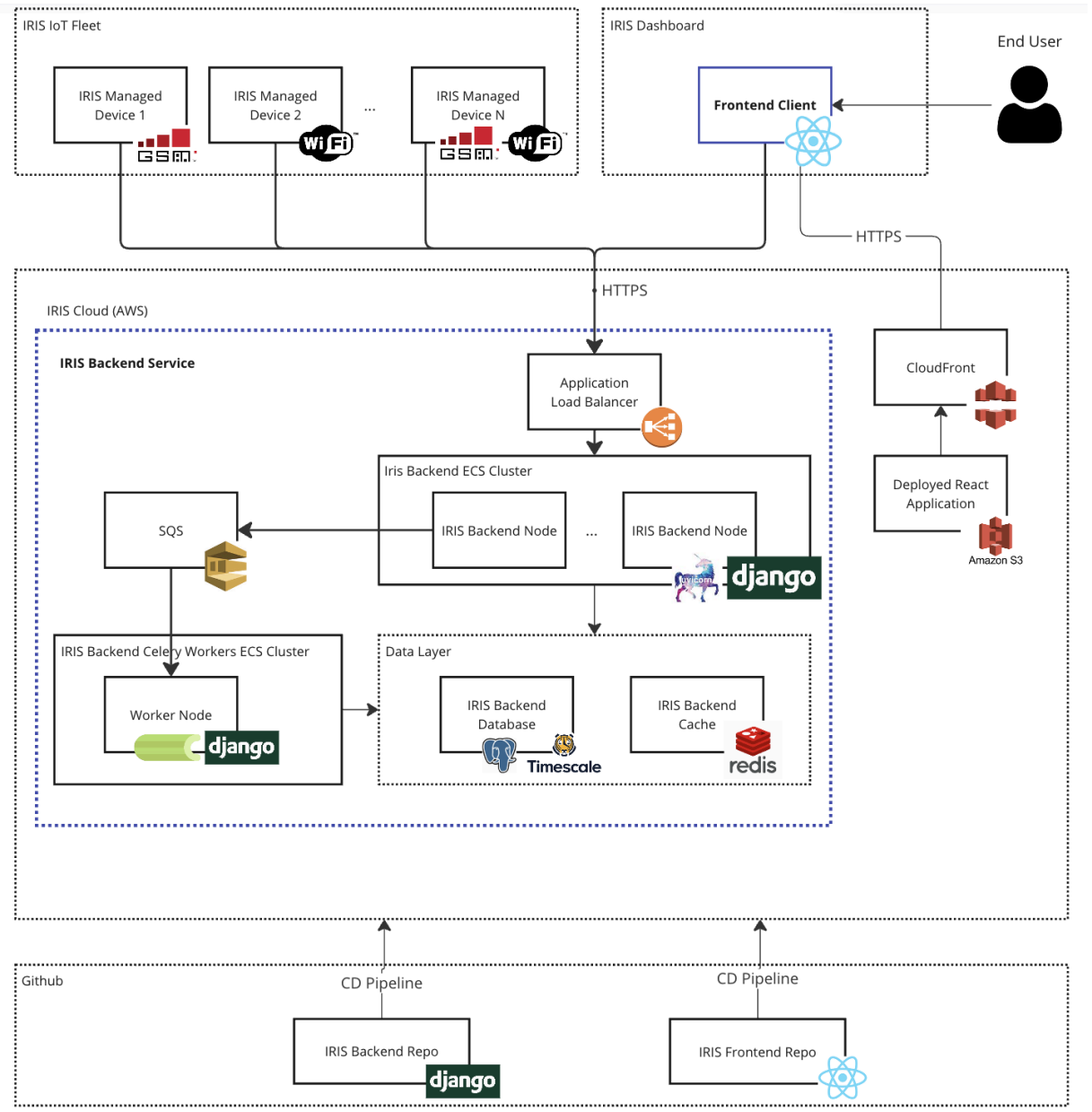
- **Segurança e identificação do equipamento:** A implementação de certificados para garantir a segurança dos dados em sistemas IoT envolve a criação e validação de certificados digitais que autenticam e autorizam dispositivos. No lado do servidor, essa gestão pode ser feita internamente, onde certificados são gerados e validados através de bibliotecas criptográficas robustas. Alternativamente, serviços integrados como AWS Certificate Manager (ACM) e AWS IoT Core podem ser utilizados, simplificando a provisão, gerenciamento e renovação de certificados, além de oferecer integração fácil com outros serviços da AWS, garantindo uma segurança escalável e confiável para dispositivos IoT.

Como implementação V0, será usado o Mac Address do equipamento como autenticação. Caso a aplicação comercial demande um requisito de segurança maior, a solução descrita acima pode ser implementada.

4.2.4 Blueprint Da Arquitetura

Com base nos resultados das etapas anteriores, é possível definir a arquitetura final da aplicação. Na figura 30, é possível entender como os componentes da nossa aplicação se conectam e as principais tecnologias utilizadas. Também será explicado o racional por trás da escolha de cada tecnologia.

Figura 30 – Visão geral da arquitetura desenvolvida



Fonte: Elaborado pelo autor (2024)

Os equipamentos se comunicam com o servidor responsável pela coleta das medições através da internet, pelo protocolo HTTPS. O servidor disponibiliza uma API na qual cada componente pode mandar uma ou um batch de medições.

O serviço IRIS Backend é responsável por processar e armazenar essas informações em banco. Posteriormente, essas informações são consumidas através de APIs pelas dashboards e disponibilizadas ao usuário final.

Os blocos destacados em azul são onde as regras de negócio da aplicação estão definidas e onde a principal carga de trabalho deste TCC se encontra. Em especial, o IRIS Backend. Este serviço:

- Mantém um registro de todos os Equipamentos e Sensores da frota.
- Mantém um registros dos Clientes e Usuários do sistema;
- Gerencia a alocação desses equipamentos nos clientes, consequentemente o acesso aos dados gerados por esses equipamentos;
- Recebe, processa e armazena os dados dos equipamentos;
- Orquestra os processos de ETL com fontes de dados externas;
- Disponibiliza os dados para o Frontend.

Como apresentado na modelagem dos domínios da aplicação, estes poderiam ser facilmente quebrados em micro serviços, seguindo o pattern de microservice-per-domain. No entanto, como definido nas premissas do projeto, um monolito Django seria mais rápido de implementar, dando mais agilidade para a frente comercial poder vender provas de conceito em clientes. De qualquer forma, o desacoplamento ainda é mantido por uma boa modelagem do software seguindo os princípios da *Clean Architecture*. Permitindo assim uma possível refatoração de microserviços estratégicos em uma evolução do projeto.

4.2.4.1 Ferramentas Utilizadas

Web Framework: O framework escolhido para o desenvolvimento do backend foi o Django 4.0. O motivo por trás da escolha se deu principalmente pelos critérios abaixo:

- Facilidade de contratação de desenvolvedores;
- Tamanho da comunidade e grande gama de ferramentas;
- Django Admin resultaria em uma interface pronta para a gestão dos equipamentos out-of-the-box. Uma descrição detalhada dos motivos pode ser encontrada no APÊNDICE C.

Banco de Dados: A escolha do paradigma relacional para o armazenamento dos dados da aplicação foi natural pela característica da modelagem dos dados, sendo PostgreSQL e TimescaleDB. Nesse contexto o PostgreSQL se destaca por sua robustez, flexibilidade e alto desempenho no processamento de dados (POSTGRESQL, 2023). Uma descrição mais detalhada da comparação entre TimescaleDB e a outra alternativa padrão de mercado, o InfluxDB pode ser encontrada no APÊNDICE D.

O maior desafio do projeto é satisfazer a necessidade de alta performance na aplicação e características específicas das queries, especialmente no que diz respeito ao processamento de dados temporais. Esse aspecto é fundamental na gestão e análise de dados ambientais coletados pelos dispositivos IoT. Nesse quesito, o TimescaleDB foi a ferramenta que se destacou nos seguintes critérios:

Performance no Processamento de Dados Temporais: TimescaleDB é uma extensão do PostgreSQL otimizada para séries temporais, oferecendo alta performance para leitura e escrita de dados temporais. Ele é capaz de lidar com grandes volumes de dados e suporta consultas complexas de forma eficiente. A arquitetura de TimescaleDB permite a compressão de dados, particionamento automático (hypertables), e otimizações específicas para consultas temporais, como agregações e ordenações (Timescale, 2023).

Integração com PostgreSQL: TimescaleDB herda todas as vantagens do PostgreSQL, incluindo suporte a ACID transactions, uma ampla gama de tipos de dados, e a capacidade de executar queries SQL padrão. Além disso, isso facilita a contratação de desenvolvedores familiarizados com PostgreSQL e simplifica a integração com outras ferramentas e bibliotecas que já suportam PostgreSQL (PostgreSQL, 2023). Esse ponto era de extrema importância para garantir celeridade no desenvolvimento, como explicado na arquitetura. A capacidade de manter a aplicação como um monolito, diminui drasticamente a complexidade da aplicação, consequentemente o tempo de entrega.

Escalabilidade e Manutenção: A arquitetura do TimescaleDB facilita a escalabilidade horizontal e vertical, permitindo a adição de mais recursos conforme necessário. A manutenção é simplificada devido à compatibilidade com as ferramentas de administração do PostgreSQL e à documentação detalhada fornecida pela Timescale (Timescale, 2023).

Ferramentas complementares: Algumas ferramentas complementares importantes de serem citadas:

- **Spatial indexing database backend:** PostGIS é uma extensão do PostgreSQL que adiciona suporte para dados geoespaciais, permitindo a execução de consultas espaciais complexas. No contexto deste projeto, PostGIS foi utilizado para indexar e gerenciar a localização geográfica das medições e dos equipamentos. Isso possibilitou o armazenamento eficiente e a consulta rápida de dados espaciais, facilitando a análise geoespacial e a visualização das posições dos dispositivos de medição no campo. Com

PostGIS, foi possível realizar operações como cálculos de distâncias, interseções e agregações espaciais, essencial para monitorar e gerenciar a distribuição dos equipamentos e suas leituras de forma precisa e em tempo real (POSTGIS, 2023).

- **Execução assíncrona e orquestramento de jobs assíncronas:** Celery
- **Django boilerplate:** django-cookiecutter
- **Conteinerização:** Docker
- **Versionamento:** Git
- **Cloud Provider:** AWS

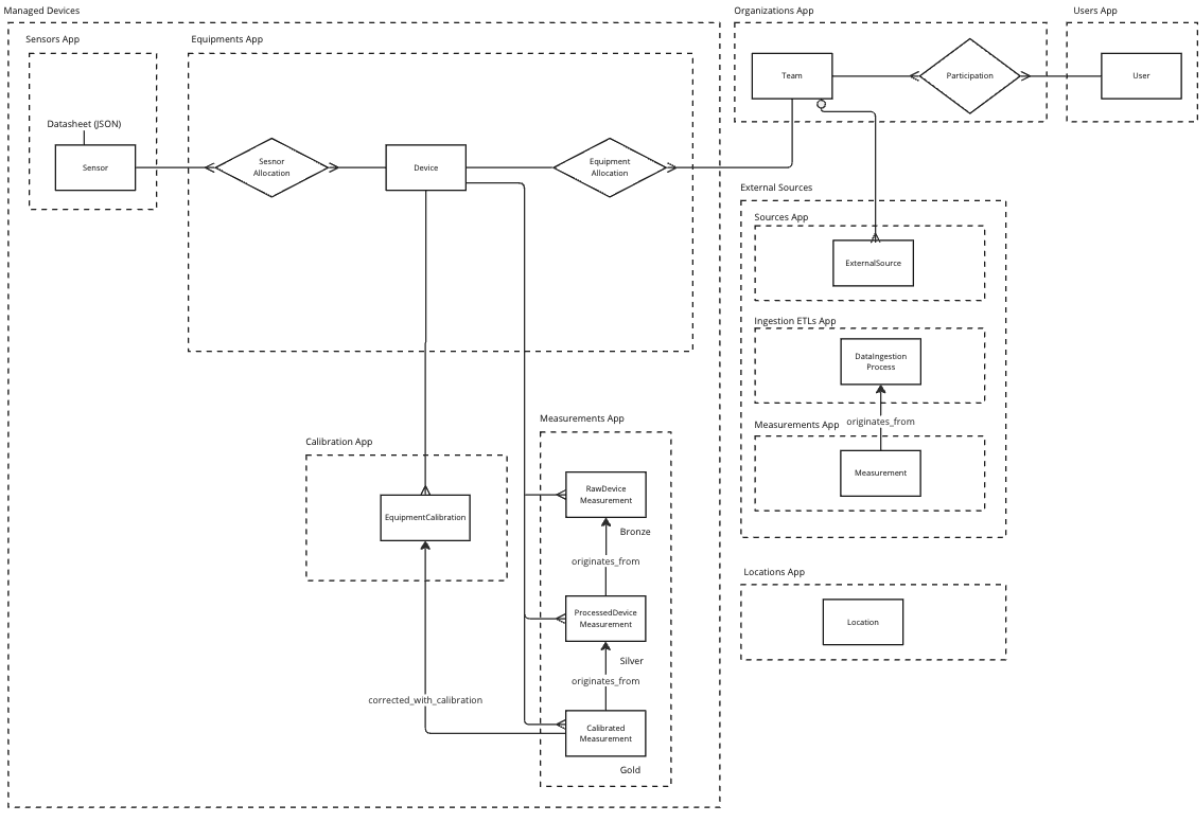
Outras ferramentas como bibliotecas, type-checkers, entre outros exemplos também foram usados, mas não são notáveis de destaque.

4.2.4.2 Arquitetura Dos Aplicativos Django

Cada domínio da modelagem é implementado por um django app respeitando os princípios SOLID e da Clean Architecture para garantir o desacoplamento dos módulos. Além disso, foram seguidos os princípios de desenvolvimento do livro: Two Scoops of Django.

O modelo de dados segue algo muito similar às entidades de cada domínio. Na figura 31 é apresentado o modelo conceitual dos dados, e os django apps no qual cada tabela é implementada.

Figura 31 - Arquitetura dos aplicativos Django



Fonte: Elaborado pelo autor (2024)

4.2.5 Planejamento e Priorização Do Escopo

No processo de desenhar a arquitetura e tomar as decisões técnicas do projeto, foi possível entender a ordem de grandeza da complexidade na implementação de cada funcionalidade. Com isso, foi chegamos no escopo priorizado que foi validado com os sócios e experts do domínio, listado no APÊNDICE E.

Por fim, com a arquitetura definida, as interfaces e regras de negócio mapeadas, e o escopo priorizado, o trabalho foi dividido em Milestones, Epics e Tasks seguindo princípios da Metodologia Ágil. Para cada Task, foi estimado um número de horas para a conclusão da tarefa. Com isso, foi possível entender o cronograma de entrega das Epics e das Milestones. As Epics eram entregas auto-contidas o suficiente para serem validadas pelo time de negócios e parceiros comerciais.

Mais informação sobre a estruturação das Epics, Tasks e como foi gerenciado o projeto está descrito no APÊNDICE F.

4.3 CONSIDERAÇÕES FINAIS DA SEÇÃO

A etapa de planejamento e arquitetura do projeto permitiu com que o desenvolvimento fosse suave. Como principais indicadores da qualidade dos resultados dessa etapa podemos citar:

- Ao todo, foram gastos aproximadamente 340 horas no desenvolvimento do projeto, enquanto o tempo gasto com tarefas classificadas como dívida técnica foi de apenas 8h. O que significa que as decisões tomadas na arquitetura permitiram que as tarefas fossem desenvolvidas sem complicações.
- Todas as Epics foram entregues no prazo, e com poucas alterações ou pontos não mapeados da definição inicial.
- Nas reuniões comerciais a apresentação da plataforma foi claramente impactante e a maior parte das requisições de funcionalidade ou já estavam contempladas, ou eram fáceis de adicionar e já suportadas pela arquitetura existente.

5 RESULTADOS - SISTEMA DESENVOLVIDO

Nesta seção serão descritos apenas os maiores desafios resolvidos na etapa de desenvolvimento e os detalhes de implementação por trás de cada um.

5.1 BACKEND

Desenvolvimento do sistema para gerenciamento da frota; e das APIs para aquisição e processamentos dos dados dos equipamentos:

5.1.1 Gestão Simplificada Da Frota

O model `Device` representa um equipamento físico construído pela IRIS. Como visto no modelo conceitual de dados, um `Device` tem um ou mais `Sensor` associados a ele. Essa associação é realizada através de uma tabela intermediária que guarda metadados dessa relação. Da mesma forma, um `Device`, quando vendido ou alugado para um cliente é alocado em uma `Organization` através de um `DeviceAllocation`. Todas as regras de negócio permeando essas relações são implementadas no models.py através de *Managers* e *Fat Models* como dita a literatura (2 scoops of Django).

Figura 32 - Implementação das regras de negócio de gestão da frota seguindo o design pattern de *Fat Models*

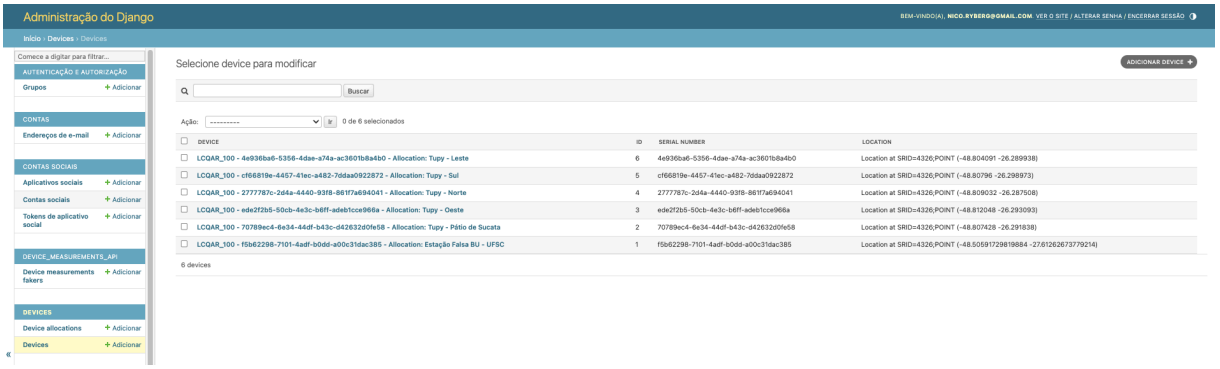
```
You, last month | 1 author (You)
class DeviceManager(models.Manager):

    def update_device_last_measurement_and_location(…

    def get_deallocated_devices(self):…


You, last month | 1 author (You)
class Device(BaseModel, LocationMixin):

    objects: DeviceManager = DeviceManager()

    sensors: "models.QuerySet[SensorAllocation]"

    # auto-incrementing primary key used for IoT payload size optimization
    _id = models.AutoField(primary_key=True)
    # serial_number is a UUID to avoid exposing the device's internal ID
    serial_number = models.UUIDField(default=uuid4, unique=True, editable=False)

    last_measurement = models.DateTimeField(null=True, blank=True)

    model = models.CharField(choices=list_to_choices(DEVICE_MODEL_CHOICES))

    def allocate_to_team(self, team_id: int, start_date: datetime | None = None):
        DeviceAllocation.objects.create(device=self, team_id=team_id, start_date=datetime_or_now(start_date))

    def deallocate(self, end_date: datetime | None = None):
        DeviceAllocation.objects.deallocate_device(self.pk, end_date)

    @transaction.atomic
    def reallocate(…

    def get_active_allocation(self) -> "DeviceAllocation | None":…

    def get_sensors(self):…

    def save(self, *args, **kwargs):…

    def attach_sensor(self, sensor: Sensor, device_slot: str):…

    def detach_sensor(self, sensor: Sensor):…

    def __str__(self):
        return f"{self.model} - {self.serial_number} - Allocation: {self.get_active_allocation()}"
```

Fonte: Elaborado pelo autor (2024)

Para garantir a consistência das informações, além de serem aplicadas proteções na camada de aplicação para que um mesmo sensor não seja alocado em dois equipamentos e que um mesmo equipamento não seja alocado em dois times, o que poderia provocar um vazamento de dados sensíveis, é implementada uma constraint no PostgreSQL.

A interface de gerenciamento utilizada pelo time da IRIS é o próprio painel de administração do Django:

Figura 33 - Interface de gestão dos equipamentos



Fonte: Elaborado pelo autor (2024)

5.1.2 Escalabilidade

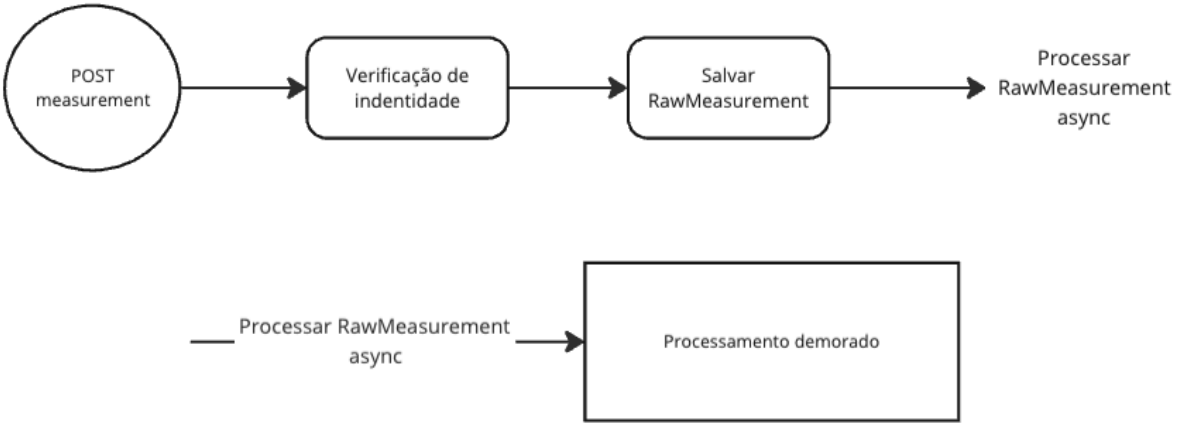
As necessidades centrais de escalabilidade foram endereçadas pela escolha correta de ferramentas e modelagem desenhada dos dados e já foram descritos nas seções anteriores. Por exemplo, o servidor web, conforme apresentado no blueprint da arquitetura, opera de maneira distribuída. Sempre que mais poder de processamento é necessário, mais containers Docker podem ser executados para escalar horizontalmente a aplicação.

Os pontos descritos abaixo são detalhes de implementação que buscam otimizar a escalabilidade da aplicação, não necessariamente respondidas na visão macro da arquitetura.

5.1.2.1 Baixa Latência De Resposta: Message Brokers E Tarefas Assíncronas

Para garantir uma baixa latência na ingestão das medidas, o processamento das medidas é delegado para uma tarefa assíncrona assim que chega, como demonstrado no diagrama abaixo. Essa delegação assíncrona foi implementada usando o Celery, e um message broker: Redis em desenvolvimento e AWS SQS em produção.

Figura 34 - Diagrama da execução assíncrona do processamento das medidas



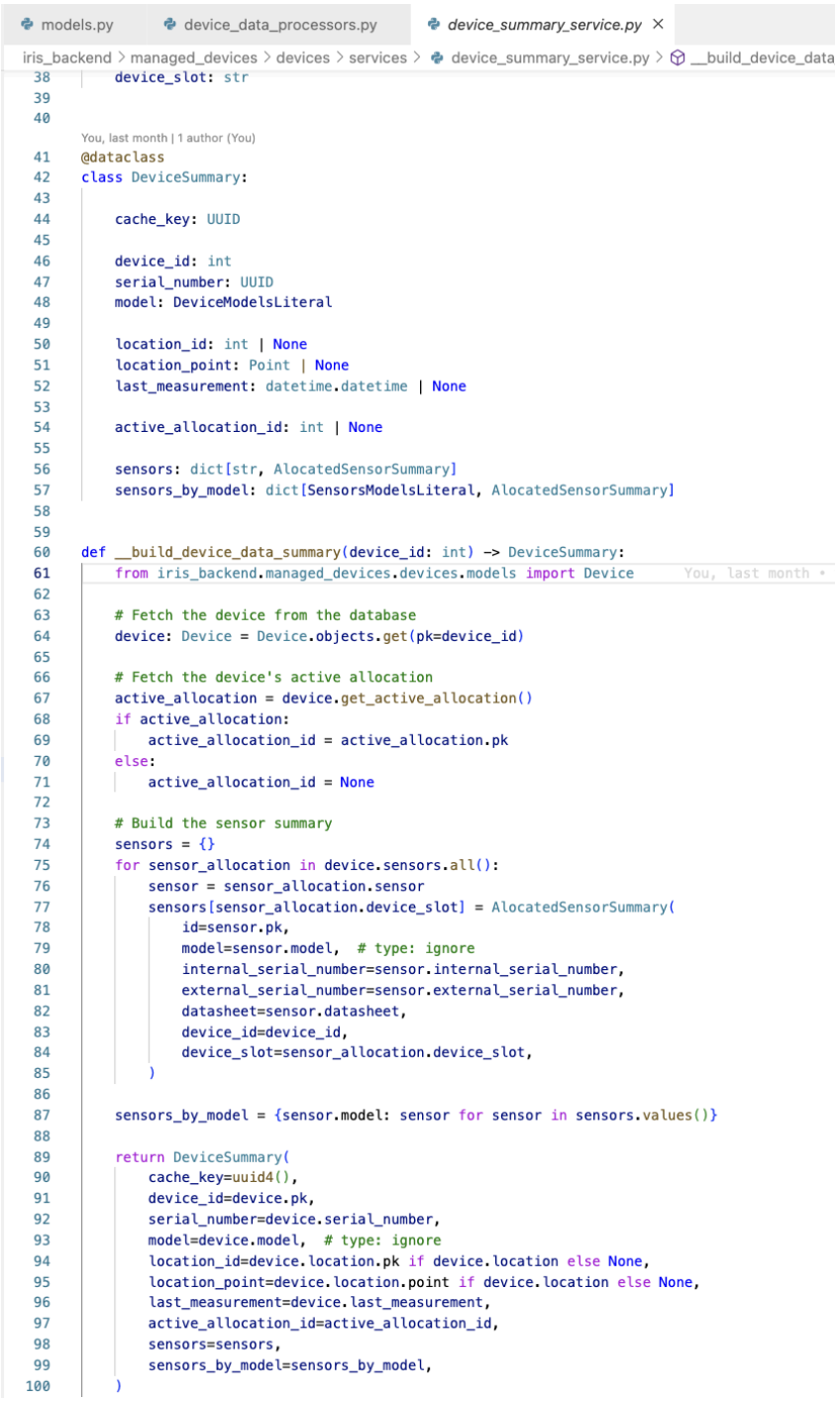
Fonte: Elaborado pelo autor (2024)

5.1.2.2 Otimização De Performance Na Inserção De Novas Medições Através De Gerenciamento De Cache Dos Equipamentos

Para coletar medidas dos equipamentos, é necessário consultar uma série de informações do equipamento, e dos sensores alocados. Visto que nossa aplicação tem um volume grande de coleta, é uma otimização importante alterar a necessidade dessas consultas no banco, por uma consulta mais eficiente em memória no cache da aplicação.

A interação com esse cache é implementada no managed_devices/devices/service/device_summary_cache.py.

Figura 35 - Implementação de um *DeviceSummary* serializável para cache



Fonte: Elaborado pelo autor (2024)

No entanto, o maior desafio lidando com gerenciamento de cache, é garantir que o cache não esteja defasado da informação armazenada no banco de dados. Desta forma, é importante implementar uma estratégia de invalidação de cache.

Na aplicação, por fins de segurança, adotamos uma estratégia de invalidação de cache bastante conservadora: qualquer UPDATE em um equipamento, ou em suas tabelas

relacionadas (DeviceAllocation, Sensor e SensorAllocation) resultaria em uma invalidação de cache. Isso foi implementado sobrescrevendo o método `save` destes modelos e no caso do modelo `Sensor`, como este não possui conhecimento sobre o seu `Device`, foi acoplada a função de invalidação de cache ao sinal emitido após mudanças no modelo.

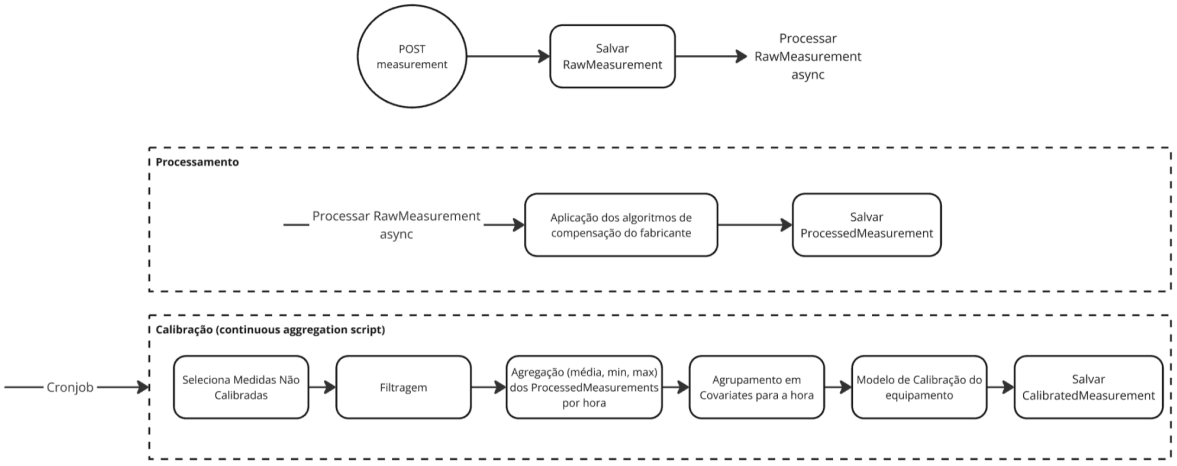
5.1.2.3 Otimização De Performance Na Consulta De Informação Por Meio Do Uso De Indexação Temporal E Continuous Aggregates

A otimização de performance na consulta de informações é crucial em sistemas com grandes volumes de dados de séries temporais. A TimescaleDB, uma extensão do PostgreSQL, oferece indexação temporal e Continuous Aggregates para melhorar a eficiência das consultas. A indexação temporal acelera a busca de dados em intervalos de tempo específicos. Já os Continuous Aggregates pré-agregam dados, permitindo consultas rápidas e reduzindo a carga computacional ao processar grandes volumes de dados em tempo real (TimescaleDB, 2021). Nas tabelas de dados Gold, consumidas pela dashboard, essas funcionalidades foram implementadas para garantir que consultas complexas sejam eficientes e em tempo hábil, facilitando a análise e a geração de relatórios sem comprometer a performance do sistema.

5.1.3 Aquisição, Processamento E Calibração Das Medidas

Foi definido que o processamento dos dados seria de exclusividade única do servidor, o equipamento sendo responsável apenas pelo envio dos dados brutos. No servidor, esse processo foi dividido em duas etapas, representadas pelo fluxograma da figura 36.

Figura 36 - Fluxograma de ponta a ponta da ingestão de um dado bruto ao armazenamento de uma variável de poluição atmosférica

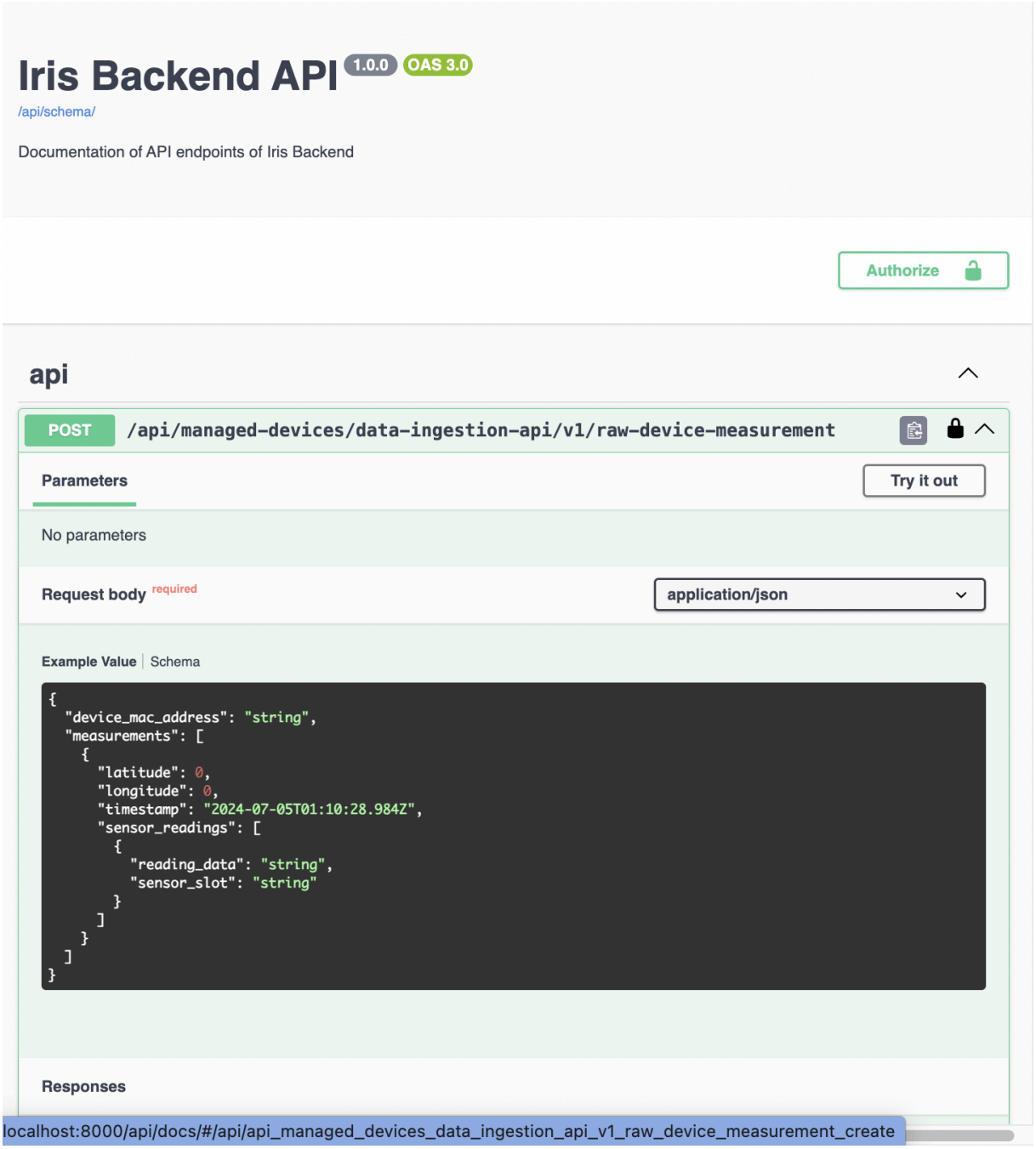


Fonte: Elaborado pelo autor (2024)

- Processamento dos dados brutos a partir dos algoritmos de compensação dos fabricantes chamada daqui pra frente de “processamento”;
- Calibração dos dados processados a partir de algoritmo de calibração proprietário da IRIS chamada daqui pra frente de “calibração”;

A API de aquisição segue a estrutura da imagem abaixo. Na qual pode receber uma ou um batch de *measurements*. A tipagem da API é definida pelo *DeviceDataProcessor* descrito através da figura 37.

Figura 37 - Documentação da API de ingestão de medidas do equipamento



Fonte: Elaborado pelo autor (2024)

5.1.3.1 Interface Genérica Para O Processamento De Dados Brutos Dos Equipamentos.

Para realizar o processamento dos dados brutos, foram criadas três interfaces: Sensor DatasheetDefinition, SensorDataProcessor e DeviceDataProcessor. Além disso, foi definido

um DTO para uma variável processada, o *ProcessedVariable*. Esse será o output de qualquer etapa de processamento de dados.

- **Primeiro passo: armazenar constantes dos sensores (*DatasheetDefinition)***

Essa informação é armazenada em um JSON Field pela natureza flexível de esquema da informação, uma vez que cada modelo de sensor tem diferentes tipos de constantes. Como esta informação pertence a cada instância de sensor, é armazenada na tabela `Sensor`.

A validação do datasheet então, não é feita na camada de dados, mas na camada de aplicação e implementada no service: sensors/service/datasheets.py.

Figura 38 - Implementação de um ExampleDatasheet

```
59  class ExampleDatasheet(DatasheetDefinition):
60
61      _MODELS = ("EXAMPLE_MODEL",)
62
63      constant_1: float
64      constant_2: float
```

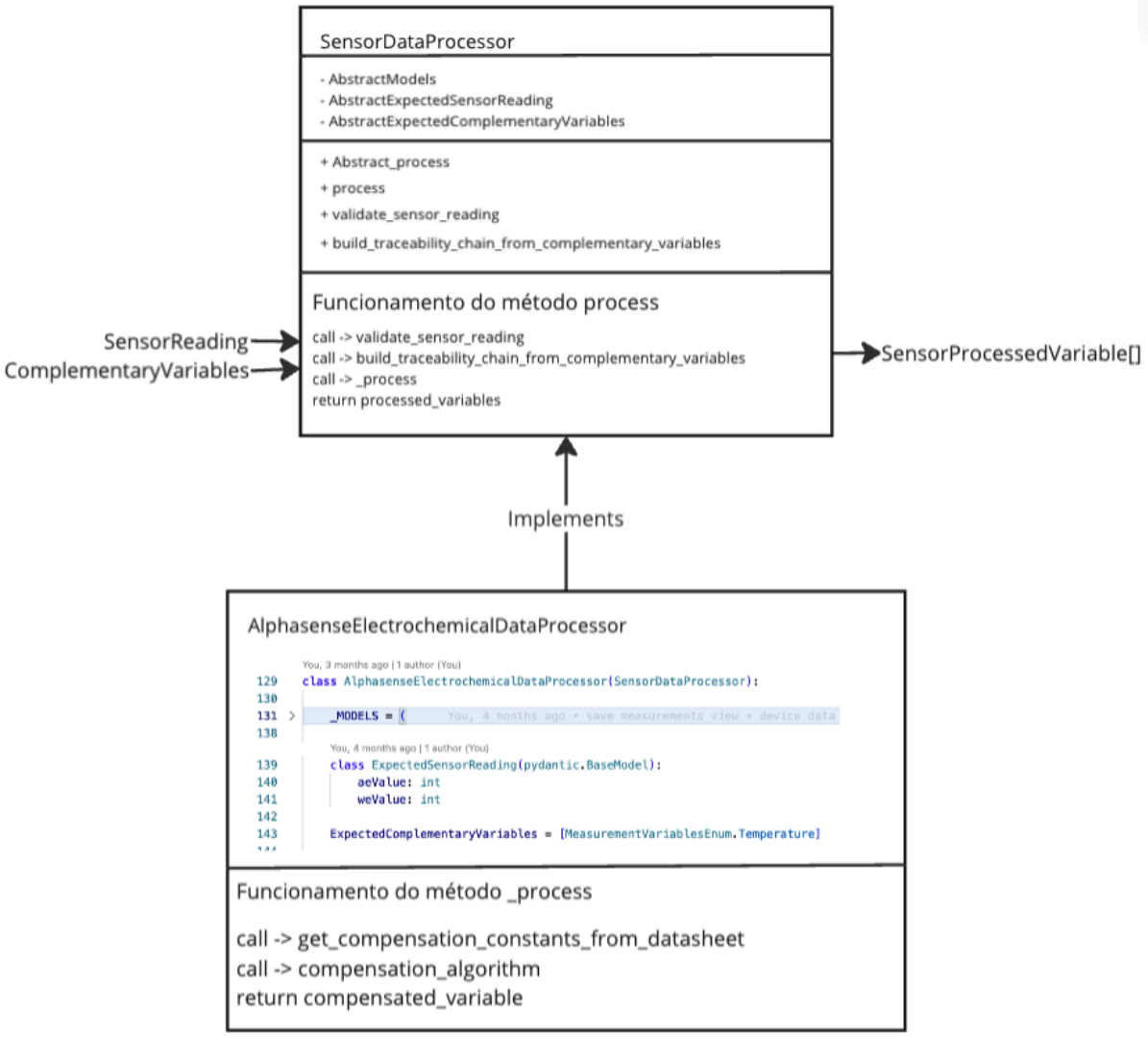
Fonte: Elaborado pelo autor (2024)

Os SensorDatasheets são modelos Pydantic que aplicam as validações necessárias para garantir que foram inseridos dados válidos para o sensor. Garantindo a consistência das informações. A interface genérica permite que o *Sensor* model faça essa validação independente da estrutura do datasheet e do modelo do sensor. Além disso, garante uma rigidez no esquema dos dados para mesmos modelos de sensores, enquanto permite flexibilidade inter-modelos.

- **Segundo passo: processar os dados brutos de um sensor (*SensorDataProcessor*)**

Um SensorDataProcessor é responsável por analisar os dados brutos da leitura do sensor e processá-los em variáveis que podem ser usadas para criar uma medição. Ele aplicará toda a lógica necessária para converter os dados brutos em variáveis, como as transformações fornecidas pela ficha técnica. Ele permite que criemos uma interface única para transformar dados brutos de um sensor em valores de concentração, aplicando as funções de compensação definidas pelo fabricante do sensor.

Figura 39 - Diagrama de funcionamento de um SensorDataProcessor



Fonte: Elaborado pelo autor (2024)

Interface: A implementação da classe SensorDataProcessor (APÊNDICE G) define uma interface que padroniza o processamento de leituras de sensores. Ele inclui métodos e atributos que devem ser implementados por classes específicas de processadores de dados de sensores. As classes derivadas devem definir:

- ExpectedSensorReading: Um modelo baseado em Pydantic que valida a estrutura e os tipos dos dados brutos recebidos do sensor.

- ExpectedComplementaryVariables: Uma lista de variáveis complementares que podem ser necessárias para o processamento dos dados do sensor.
- _process: O método abstrato _process recebe duas entradas principais: sensor_reading, que é um modelo Pydantic contendo a leitura bruta do sensor validada e tipificada (por exemplo, valores de eletrodos como aeValue e weValue), e complementary_variables, uma lista opcional de variáveis complementares (SensorProcessedVariable) necessárias para o processamento, como a temperatura ambiente. A saída do método é uma lista de SensorProcessedVariable, cada uma representando uma variável processada, incluindo o valor calculado e a cadeia de IDs dos sensores utilizados no processamento, garantindo a rastreabilidade dos dados transformados.

Funcionamento: Ao receber uma leitura bruta do sensor, o processador valida os dados utilizando o modelo ExpectedSensorReading. Se os dados forem válidos, ele os processa usando o método _process, que aplica as transformações e cálculos definidos para aquele tipo de sensor. O resultado é uma lista de SensorProcessedVariable, que inclui as variáveis processadas e a cadeia de IDs dos sensores utilizados no processamento, garantindo a rastreabilidade dos dados.

Garantia de Rastreabilidade da Informação: A rastreabilidade é garantida através do atributo sensor_ids_chain presente na classe *SensorProcessedVariable*. Este atributo armazena a cadeia de IDs dos sensores que contribuíram para o processamento da variável, permitindo que se trace a origem e as transformações aplicadas aos dados ao longo do tempo. Isso é crucial para garantir a integridade e a confiabilidade das medições.

Exemplo para sensor eletroquímico da Alphasense: A classe *AlphasenseElectrochemicalDataProcessor* é um exemplo específico de um processador de dados de sensor eletroquímico. Ela processa leituras de sensores que medem concentrações de gases como CO, CO2, NO2, SO2 e O3. Esta classe implementa o método *_process* para aplicar a lógica de compensação específica desses sensores, considerando variáveis complementares como a temperatura ambiente.

- *ExpectedSensorReading*: Define que a leitura do sensor deve conter valores aeValue e weValue.

- *ExpectedComplementaryVariables*: Especifica que a temperatura é uma variável complementar necessária.
- *_process*: Este método aplica um algoritmo de compensação utilizando os valores de aeValue e weValue, a temperatura e outros parâmetros do datasheet do sensor, como sensibilidade e fatores de correção. O resultado é uma variável processada que representa a concentração do gás detectado pelo sensor.

Em resumo, o *SensorDataProcessor* e suas subclasses proporcionam uma estrutura modular e extensível para o processamento de dados de sensores, garantindo a validação, transformação e rastreabilidade das informações coletadas, o que é essencial para a precisão e confiabilidade das medições em aplicações de monitoramento ambiental.

- **Terceiro passo: acoplar as medições de diferentes sensores (*DeviceDataProcessor*)**

O processador de dados do dispositivo (DeviceDataProcessor) é responsável por analisar os dados brutos recebidos do dispositivo e processá-los em variáveis que podem ser usadas para criar medições. O processador lê os dados dos sensores na ordem correta, garantindo que cada processador de sensor seja chamado com as variáveis esperadas. Por exemplo, um sensor de CO no dispositivo espera que a temperatura seja processada primeiro. A implementação dessa classe pode ser encontrada no APÊNDICE H.

A interface implementada é de certa forma similar a de um `SensorDataProcessor`. Além disso, garante a coesão e o acoplamento entre diferentes sensores independentes. Desta forma a implementação dos sensores pode ser desenvolvida de forma desacoplada. Por exemplo, um `SensorDataProcessor` que depende da temperatura para aplicar o algoritmo de compensação, poderia pegar essa temperatura de qualquer sensor que forneça essa informação, independente de como a estrutura do dado bruto foi fornecida. O mecanismo responsável por definir essa interface e garantir a coesão é o `DeviceDataProcessor`.

O `DeviceDataProcessor` implementa modelos de equipamento, por exemplo o LCQAR_100. Desta forma, se mudanças críticas na composição de sensores fossem realizadas, uma nova versão do equipamento deverá ser lançada. Esse requisito nos traz algumas bases para o versionamento de equipamentos no futuro.

Figura 40 - Implementação do LCQAR_100_DataProcessor

```python
You, last month | 1 author (You)
class LCQAR_100_DataProcessor(DeviceDataProcessor):
    _MODELS = ["LCQAR_100"]

    You, last month | 1 author (You)
    class ExpectedSensorReadings(pydantic.BaseModel):

        TEMPERATURE_SENSOR: ExampleTemperatureDataProcessor.ExpectedSensorReading
        ALPHASENSE_CO: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None
        ALPHASENSE_CO2: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None
        ALPHASENSE_NO2: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None
        ALPHASENSE_SO2: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None
        ALPHASENSE_O3: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None
        ALPHASENSE_MP: AlphasenseElectrochemicalDataProcessor.ExpectedSensorReading | None = None

    You, 3 months ago | 1 author (You)
    class ExpectedDeviceMeasurement(pydantic.BaseModel):
        location: PydanticPointType | None = None
        timestamp: datetime.datetime
        sensor_readings: "LCQAR_100_DataProcessor.ExpectedSensorReadings"

    def _process_temperature(
        self, temperature_sensor_reading: ExampleTemperatureDataProcessor.ExpectedSensorReading      You, 3 months
    ) -> SensorProcessedVariable:
        try:
            temperature_sensor = self.device_summary.sensors_by_model["TEMPERATURE_SENSOR"]
        except KeyError:
            raise ValueError("No temperature sensor configured for device.")

        temperature_sensor_data_processor = get_sensor_data_processor(
            "TEMPERATURE_SENSOR", temperature_sensor  # type: ignore
        )
        temperature_variable = temperature_sensor_data_processor.process(temperature_sensor_reading)[0]
        return temperature_variable

    def _process_alphasense_electrochemical_sensors_reading_data(
        self, device_measurement: ExpectedDeviceMeasurement, temperature_variable: SensorProcessedVariable
    ) -> list[SensorProcessedVariable]:

        ELECTROCHEMICAL_SENSORS: list[SensorsModelsLiteral] = [
            "ALPHASENSE_CO",
            "ALPHASENSE_CO2",
            "ALPHASENSE_NO2",
            "ALPHASENSE_SO2",
            "ALPHASENSE_O3",
        ]

        processed_variables = []

        for sensor_model in ELECTROCHEMICAL_SENSORS:
            sensor_summary = self.device_summary.sensors_by_model.get(sensor_model)
            sensor_reading = getattr(device_measurement.sensor_readings, sensor_model, None)
            if sensor_summary is None:
                if sensor_reading is not None:
                    raise ValueError(
                        f"Sensor {sensor_model} is not configured for device, but a reading was provided."
                    )
                continue

            if sensor_reading is None:
                continue

            sensor_data_processor = get_sensor_data_processor(sensor_model, sensor_summary)  # type: ignore
            sensor_processed_variables = sensor_data_processor.process(sensor_reading, [temperature_variable])
            processed_variables.extend(sensor_processed_variables)

        return []

    def _process(self, device_measurement: ExpectedDeviceMeasurement) -> list[SensorProcessedVariable]:
        temperature = self._process_temperature(device_measurement.sensor_readings.TEMPERATURE_SENSOR)
        electrochemical_sensors_variablees = self._process_alphasense_electrochemical_sensors_reading_data(
            device_measurement, temperature
        )

        return [temperature] + electrochemical_sensors_variablees
```

Fonte: Elaborado pelo autor (2024)

Por fim, pode-se esconder um processo extremamente complicado atrás de uma interface simples e que funciona para qualquer composição de sensores e modelos diferentes de sensores.

Figura 41 - Implementação do processamento das medidas brutas de um equipamento de forma genérica por um DeviceDataProcessor

```python
def process_raw_measurement_v1(
    raw_measurement: RawDeviceMeasurement, device_summary: device_summary_service.DeviceSummary
):
    device_data_processor = device_data_processors.get_device_data_processor(device_summary.model, device_summary)

    sensor_readings = [sensor_reading for sensor_reading in raw_measurement.raw_data]

    sensor_readings_by_model = {}
    for sensor_reading in sensor_readings:
        sensor_readings_by_model[device_summary.sensors[sensor_reading.sensor_id].model] = sensor_reading.reading_data

    processed_variables = device_data_processor.process(
        {
            "location": point_as_tuple(raw_measurement.location) if raw_measurement.location else None,
            "timestamp": raw_measurement.time,
            "sensor_readings": sensor_readings_by_model,
        }
    )

    return save_measurements_from_processed_variables(raw_measurement, processed_variables)
```

Fonte: Elaborado pelo autor (2024)

5.1.3.2 Etapa De Calibração: Implementação De Uma Prova De Conceito De Calibração.

A metodologia de calibração da IRIS está em desenvolvimento, e no momento de construção deste trabalho, ainda não possui suas regras de negócio definidas, como por exemplo, thresholds da etapa de filtragem, modelo usado, ou o próprio pipeline de transformação dos dados. Desta forma, qualquer tentativa de criação de uma interface generalista seria uma completa perda de tempo. Portanto, foi criado um script direto para validar o conceito de calibração em campo por modelo de aprendizado de máquina.

Figura 42 - Pipeline com as etapas de calibração

Fonte: Elaborado pelo autor (2024)

A calibração segue o pipeline mostrado na figura 40. Opera na estrutura de um *continuous aggregation*, no qual as ferramentas de indexação temporal contribuem significativamente para a performance dessas agregações.

Um cronjob é executado de forma horária para calibrar todas as ProcessedMeasurements geradas na etapa anterior que ainda não foram calibradas. Esta seleção é feita através do último timestamp da medida calibrada, pela função:

Figura 43 - Implementação da seleção de medidas para continuous aggregation

```
66  def get_uncalibrated_measurements_qs() -> TimescaleManager  | None:
67      """Gets the uncalibrated measurements from the last calibration time to now."""
68      last_calibration = (
69          CalibratedDeviceMeasurement.objects.last() or ProcessedDeviceMeasurement.objects.first()
70      )
71      if last_calibration:
72          last_calibration_time = last_calibration.time
73
74      now = timezone.now()
75
76      if last_calibration_time > now - timedelta(hours=1):
77          return
78
79      return ProcessedDeviceMeasurement.timescale.filter(
80          timestamp__gte=last_calibration_time,
81          timestamp__lt=now
82      )
```

Fonte: Elaborado pelo autor (2024)

Com as medidas candidatas a calibração selecionadas, é feito uma filtragem por um modelo linear de thresholding para cada variável e device. Esta definição dos limites pode ser uma configuração do CalibrationModel no futuro. No entanto, essa filtragem poderia ser uma ferramenta mais sofisticada, como um modelo de aprendizado de máquina. Nesse caso, um refactor seria necessário para delegar a responsabilidade da filtragem do PostgreSQL para o Python, como é feito no penúltimo item da calibração.

Figura 44 - Implementação do modelo de filtragem por tresholding linear

```python
def filter_valid_processed_measurements(queryset: TimescaleManager) -> TimescaleManager:
    # Initialize an empty Q object
    query = Q()

    # Get distinct device_id and variable_id pairs
    device_variable_pairs = queryset.values('device_id', 'variable_id').distinct()

    for pair in device_variable_pairs:
        device_id = pair['device_id']
        variable_id = pair['variable_id']
        threshold = get_threshold(device_id, variable_id)
        query |= Q(
            device_id=device_id,
            variable_id=variable_id,
            value__gte=threshold.min,
            value__lte=threshold.max
        )

    return queryset.filter(query)
```

Fonte: Elaborado pelo autor (2024)

Na sequência as medidas são agregadas. Como os modelos de calibração usam os valores de diferentes concentrações para corrigir o problema da correlação cruzada, as medidas das variáveis para o mesmo bucket de horário e equipamento são agrupadas para sair de uma estrutura onde cada variável é uma linha, para a estrutura onde cada variável é uma coluna. Com isso, as medidas de todas as variáveis geradas pelo equipamento, agregadas para o time bucket definido, compõem uma linha. Essa linha é o que servirá de input para o modelo de calibração.

Figura 45 - Implementação da agregação e combinação das covariantes.

```python
hourly_measurements_qs = (
    filtered_measurements_qs.time_bucket("time", "1 hour")
        .values("bucket", "device_id", "variable_id")
        .annotate(models.Avg("value"), models.Min("value"), models.Max("value"), models.Count("id"), models.Min("time"), models.Max("time"))
)

combined_hourly_measurements = combine_same_device_and_bucket_variables(hourly_measurements_qs)
```

Fonte: Elaborado pelo autor (2024)

Por fim, com os inputs para os modelos de calibração de cada equipamento montados, é chamada a função de calibração e salvos os *CalibratedMeasurements*. Por hora, um *DummyModel* está sendo usado para a calibração, pronto para ser alterado por um modelo de

calibração real de um equipamento no momento em que este fique pronto. As interfaces se manterão neste caso.

Figura 46 - Implementação da predição pelo modelo de calibração

```
168     calibrated_measurements = []
169     for device_id, hourly_data in combined_hourly_measurements.items():
170         for bucket, variable_data in hourly_data.items():
171             for i, variable in enumerate(variable_data):
172                 covariates = variable_data[:i] + variable_data[i+1:]
173                 calibrated_value = calibrate_value(device_id, variable, covariates)
174 
175                 calibrated_measurements.append(
176                     CalibratedDeviceMeasurement(
177                         device_id=variable.device_id,
178                         variable_id=variable.variable_id,
179                         value=calibrated_value,
180                         time_start=variable.min_time,
181                         time=variable.bucket
182                     )
183                 )
184 
185     created_measurements = CalibratedDeviceMeasurement.objects.bulk_create(calibrated_measurements)
186     logging.info(f"Created {len(created_measurements)} calibrated measurements")
187 
188 def calibrate_value(device_id, variable, covariates):
189     # Use the dummy model to calibrate
190     return dummy_model.predict([variable.avg_value])[0]
```

Fonte: Elaborado pelo autor (2024)

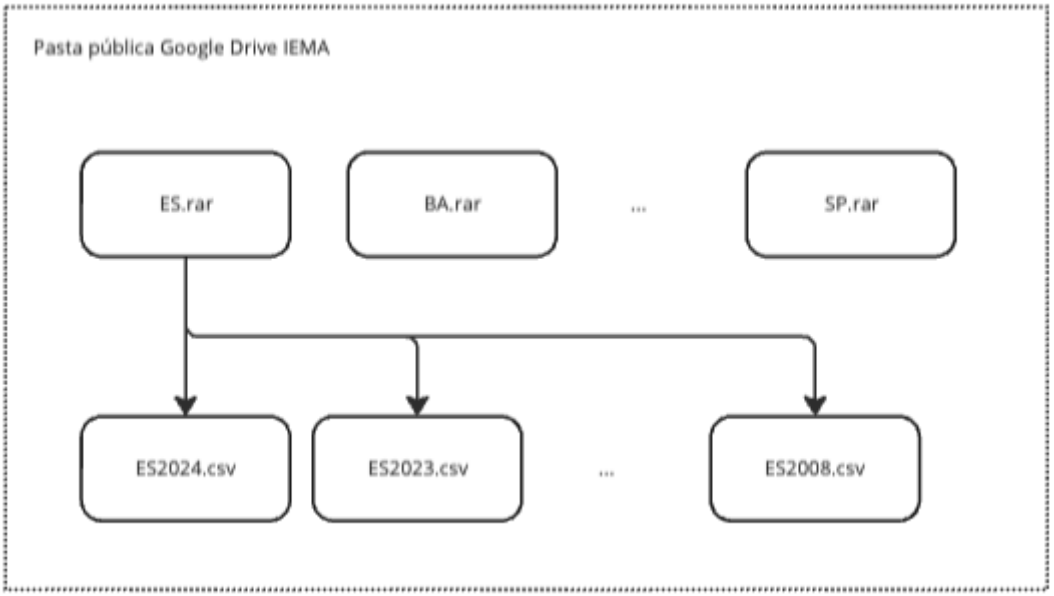
5.1.4 Integração Com Bases Externas

Para entender a solução implementada para a integração com bases externas, vamos primeiro abordar como foi implementada a ETL dos dados do MonitorAr/IEMA. A partir dessa implementação específica, entenderemos quais interfaces e ferramentas foram criadas para implementar uma solução simplificada de ETLs que possa ser aplicada em diferentes bases externas.

5.1.4.1 ETL: IEMA

Os dados do IEMA são disponibilizados em um Google Drive como arquivos .rar para cada estado, como exemplificado na figura abaixo:

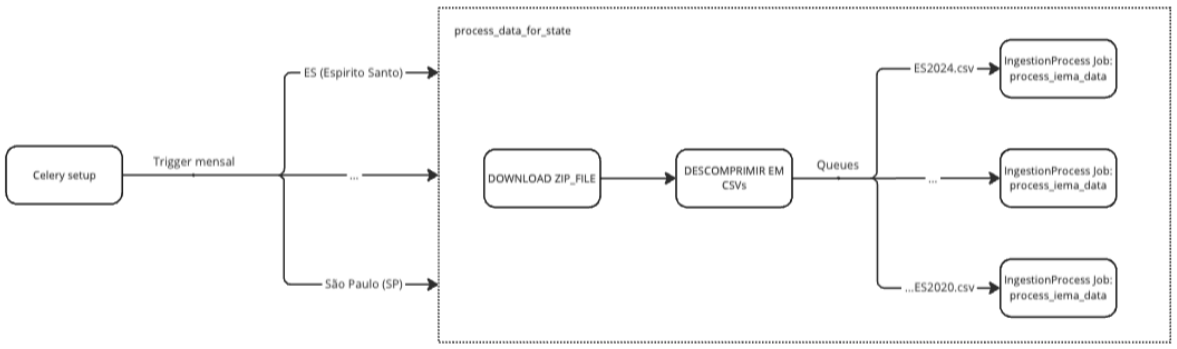
Figura 47 - Estrutura dos arquivos disponibilizados pelo IEMA



Fonte: Elaborado pelo autor (2024)

Todos os arquivos CSV seguem a mesma estrutura, porém são arquivos grandes e pesados. Desta forma, realizar esta ingestão em um único processo síncrono seria moroso. Desta forma, o processamento de cada arquivo foi quebrado em tarefas assíncronas como diagramado abaixo:

Figura 48 - Fluxograma da delegação de tarefas assíncronas de IngestionProcess do IEMA



Fonte: Elaborado pelo autor (2024)

Este é um tema recorrente na criação de ETLs e muitas ferramentas já implementam esse tipo de solução: quebrar um processo grande de ingestão em muitos pequenos processos de ingestão. Para isso criamos a classe IngestionProcess, explicada na próxima seção. O

restante das funções eram transformações específicas para o contexto dos dados do IEMA, como tratamento de dados inválidos e transformação dos dados.

5.1.4.2 Solução simplificada de ETLs

Desta forma, a solução simplificada de ETLs criada foi a seguinte: cada ETL define sua periodicidade ao cadastrar um cron job no sistema. Isso é realizado usando a ferramenta do celery beat. Esta função agendada pode gerar processos de ingestão a partir do context_manager *IngestionProcess* responsável por despachar a função de forma assíncrona, e criar um registro no banco para monitorar o status do processo de ingestão.

Figura 49 - Implementação de um IngestionProcess dispatcher

```python
@contextmanager
def ingestion_process_context(id: str, origin: SourceOriginsLiteral):
    ingestion_process = DataIngestionProcess.objects.create(
        identifier=id, status="PROCESSING", origin="IEMA", started_at=timezone.now()
    )

    try:
        yield ingestion_process
    except Exception as e:
        ingestion_process.status = "FAILED"
        ingestion_process.message = str(e)
    else:
        ingestion_process.status = "SUCCESS"
    finally:
        ingestion_process.save()


@celery.shared_task
def dispatch_async_ingestion_process(func: Callable[[DataIngestionProcess],], id: str, origin: SourceOriginsLiteral):
    with ingestion_process_context(id, origin) as ingestion_process:
        func(ingestion_process)

```

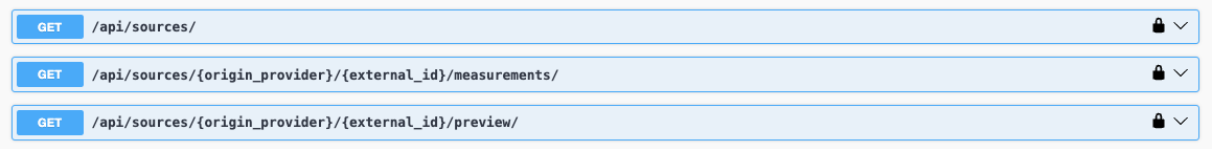
Fonte: Elaborado pelo autor (2024)

5.2 DASHBOARD

A principal premissa na priorização do escopo do dashboard foi a de evitar a criação de funcionalidades e visualizações prematuramente. Desta forma, o objetivo deste dashboard era ter uma prova de conceito para apresentar nas reuniões comerciais, sempre com essa expectativa alinhada com o cliente. Como as principais regras de negócio na modelagem dos dados já tinham sido abordados, o desenvolvimento da dashboard foi simplificado.

Foi desenvolvido um *backend-for-frontend* no próprio serviço do Django para implementar a API de consumo de medidas usadas na dashboard. Essa API expõe os seguintes endpoints, e aplica as regras de negócio para que o frontend não precise saber a

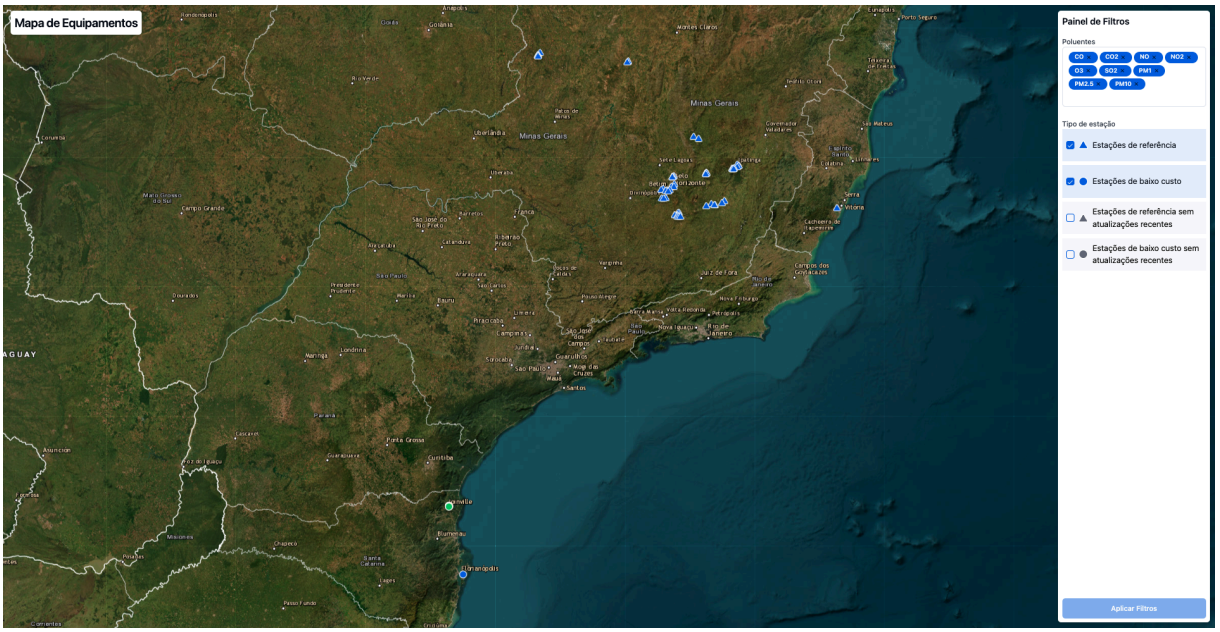
diferença entre equipamentos gerenciados pela IRIS e fontes externas. Essa regra de negócio seria bastante mais simples se fosse implementada a refatoração explicada no APÊNDICE H (futuras refatorações e aprendizados da arquitetura inicial), no tópico: uma outra forma de modelar o problema da consolidação entre Managed Devices e External Sources.



Fonte: Elaborado pelo autor (2024)

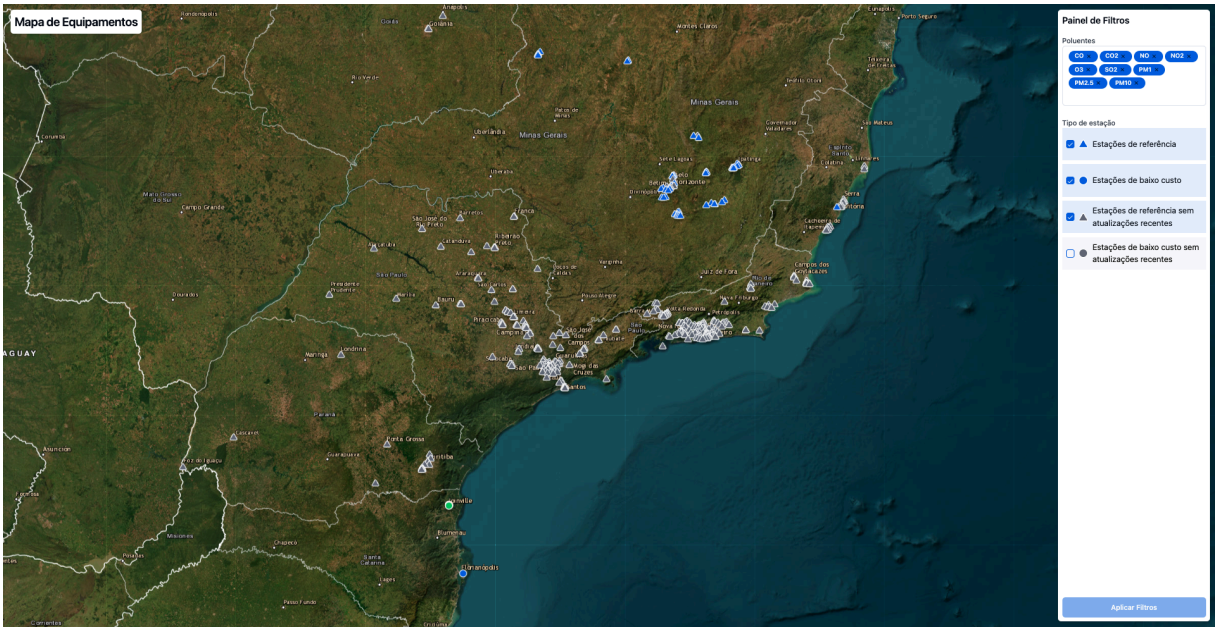
O consumo dessa API é realizado por um front em React, com duas telas: uma visão espacial das estações e uma visão de detalhe. O componente implementado mais complexo foi um painel de filtragem genérico que permitisse a fácil adição de novas dimensões de filtragem.

Figura 50 - Tela de mapa dos equipamentos na plataforma desenvolvida



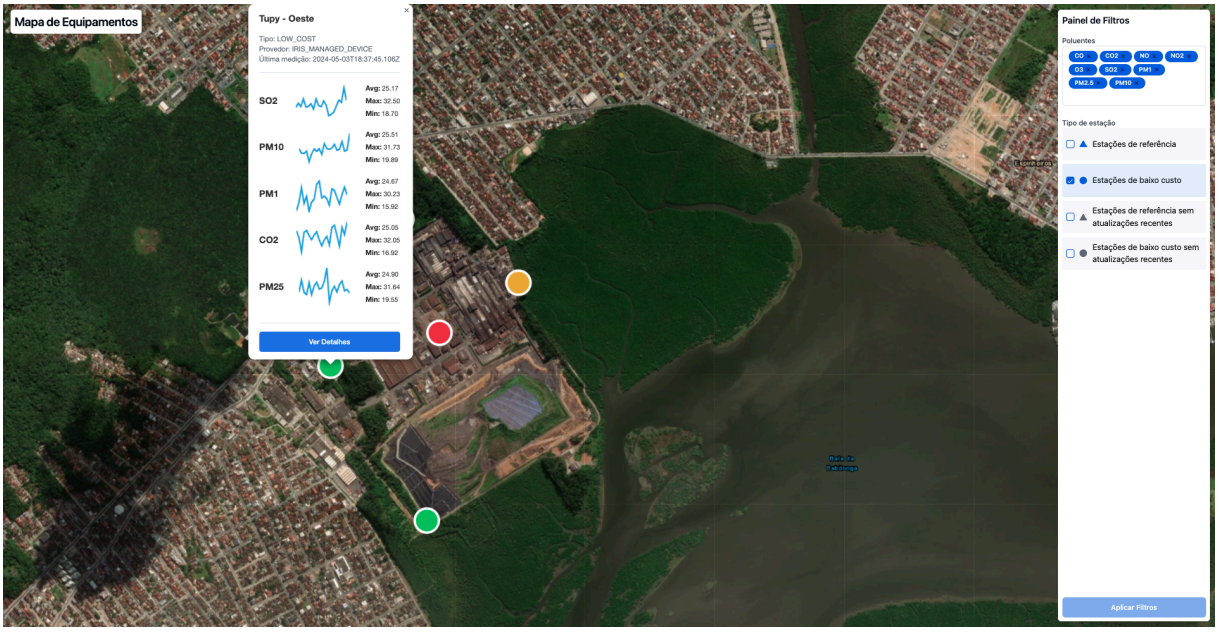
Fonte: Elaborado pelo autor (2024)

Figura 51 - Funcionamento da filtragem na plataforma desenvolvida



Fonte: Elaborado pelo autor (2024)

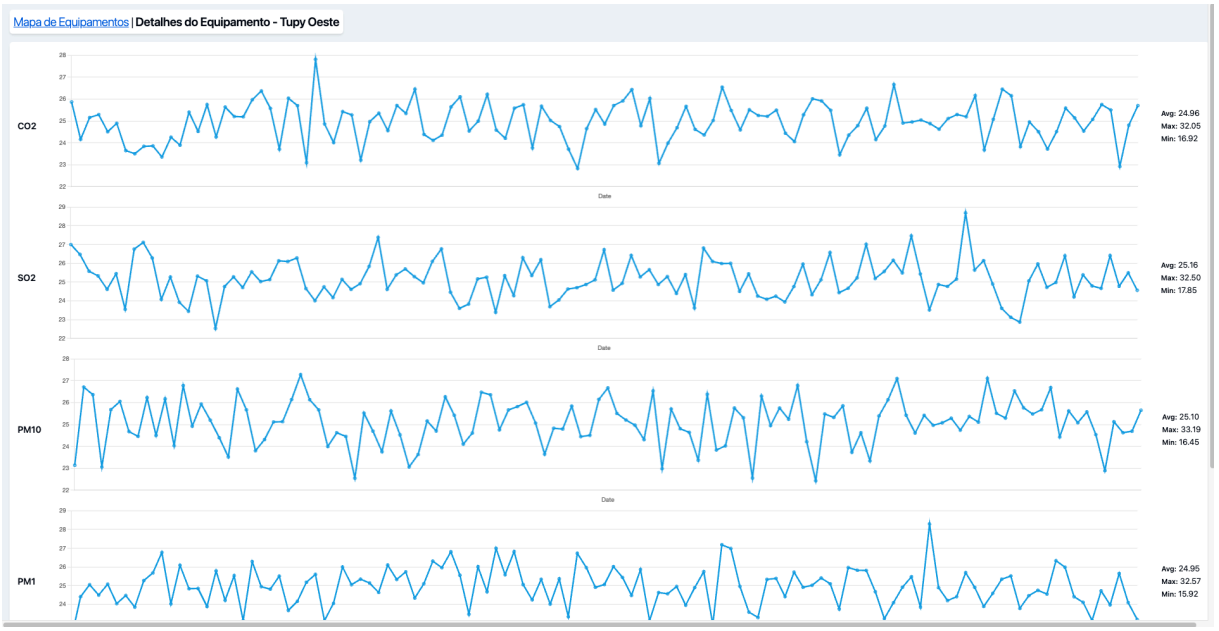
Figura 52 - Exemplo da prova de conceito apresentada para a empresa Tupy na plataforma desenvolvida



Fonte: Elaborado pelo autor (2024)

A coloração das estações foi hardcoded no código para exemplificar uma funcionalidade mapeada para o futuro de definição de limites de medição de poluentes.

Figura 53 - Tela de detalhe de estação na plataforma desenvolvida



Fonte: Elaborado pelo autor (2024)

Um ponto de melhoria na implementação atual seria disponibilizar os enumeradores e tipagem dos dados via API para o frontend. Essa refatoração também é explicada no Apêndice H (futuras refatorações e aprendizados da arquitetura inicial), no tópico de sincronização de enumeradores entre os serviços.

5.3 TESTES DE INTEGRAÇÃO DO SISTEMA

A fim de testar o processo de ponta a ponta e também facilitar a demonstração do sistema, foi criado um "simulador" de equipamento. Este sendo um cron job que gera os dados com uma periodicidade definida. Existem dois modos de simulação:

1. Simulador real de equipamento que se comporta exatamente como um equipamento, chamando a API de ingestão com dados brutos gerados.
2. Simulador de dados tratados, que gera variáveis com uma distribuição normal configurada. Essa configuração é mais simples para fazer o setup de demonstrações para os clientes.

Desta forma, é possível testar as funcionalidades e atualizações do sistema com uma frota "virtual" que se comporta como a frota real.

Em relação aos testes unitários, foram seguidos princípios de test driven development nas principais camadas de serviço da aplicação:

- sensors
    - service/datasheets.py;
    - service/sensor_data_processor.py
- devices
    - models.py (testa as constraints de alocação de equipamentos e sensores);
    - service/device_summary.py (testa os mecanismos de invalidação de cache);
    - service/device_data_processor.py.
- device_measurements_api
    - service/save_measurements.py (testa o seguimento da pipeline de processamento dos dados);
- ingestion_etls
    - iema_ingestion.py (garante que os dados processados são válidos para arquivos escolhidos arbitrariamente).

5.4 DEPLOYMENT

Os serviços foram implantados na AWS e separados em dois ambientes: homologação e produção. No entanto, a implantação do ambiente de produção será realizada apenas após a integração do primeiro equipamento à rede para controle de custos.

Stack de homologação:

- Servidor EC2 com docker-compose para rodar todos os serviços.

Stack de produção:

- ECS cluster + ALB para os serviços: django, celeryworker e celerybeat.
- RDS para o banco de dados;
- AWS ElastiCache para o cache;
- AWS SQS como message broker;
- O build do frontend é enviado para um S3 e disponibilizado através de uma rede de distribuição de conteúdo, AWS Cloudfront;

6 CONCLUSÃO

Os objetivos gerais e específicos foram concluídos com sucesso. O sistema desenvolvido está alinhado aos objetivos de curto, médio e longo prazo da IRIS.

Foi construído de maneira escalável e consegue: realizar o gerenciamento da frota de IoT da empresa; realizar a aquisição, processamento e armazenamento da medições dos equipamentos de baixo custo desenvolvidos pela empresa; integrar os dados de bases externas; e disponibilizar uma dashboard para a visualização dos dados. Além de apresentar uma arquitetura que permita a adição das funcionalidades esperadas no futuro sem grandes retrabalhos.

O sistema desenvolvido se destaca das soluções de mercado, principalmente ao introduzir uma arquitetura para o processamento e calibração dos dados em tempo real; na capacidade de integração com fontes de estações de referência, aspecto crucial na construção de sistemas de monitoramento híbridos; e aliando essas funcionalidades a requisitos robustos de segurança e acesso às informações.

A capacidade de apresentar um produto funcional foi um ponto pivotal nas reuniões comerciais da IRIS no primeiro semestre. O que permitiu que a empresa apresentasse soluções tangíveis aos clientes, como foi o exemplo da Tupy, ilustrado na seção 5.3.

Na condição atual, o produto permite que a empresa comece a operar e ganhar maturidade no desenvolvimento de redes de monitoramento da qualidade do ar. Com essa experiência, será possível implementar as evoluções já mapeadas para o produto, ao mesmo tempo que a empresa recebe feedback dos clientes.

O próximo passo imediato do projeto é integrar os primeiros equipamentos da IRIS na plataforma. Essa integração, está pendente da finalização do novo projeto eletrônico, desenvolvido em paralelo a este trabalho.

Dentre os requisitos despriorizados, podemos citar como principais evoluções futuras deste trabalho:

1. Aprimoramento das funcionalidades em torno da calibração: capacidade de orquestrar e gerenciar os modelos de calibração do equipamento; continuamente desenvolver esses modelos na aplicação; técnicas integradas de análise de dados e identificação de falhas; e dashboards para dar transparência ao usuário final sobre a confiabilidade do modelo.

2. Integração das bases de dados públicas existentes. Não só de estações de referência e baixo custo, mas também de: dados de satélite, inventário de emissões e modelagem da qualidade do ar;
3. Evolução na conectividade, independência e resiliência dos equipamentos. Permitindo a instalação em locais remotos, principalmente por meio da: implementação de uma célula de bateria/solar para alimentação do equipamento; inclusão de internet satelital no equipamento.
4. Criação de novas visualizações e evolução da capacidade de extrair insights da dashboard.

REFERÊNCIAS

AMAZON REDSHIFT. Disponível em: https://aws.amazon.com/redshift/. Acesso em: 5 jul. 2024.

AMAZON WEB SERVICES (AWS). Disponível em: https://aws.amazon.com/. Acesso em: 5 jul. 2024.

ATZORI, Luigi et al. **The Internet of Things: a survey**. Computer Networks, [S.L.], v. 54, n. 15, p. 2787-2805, out. 2010. Elsevier BV. http://dx.doi.org/10.1016/j.comnet.2010.05.010.

BECK, K.; BEEDLE, M.; VAN BENNEKUM, A.; et al. **Manifesto for Agile Software Development**. 2001.

BONOMI, F.; MILITO, R.; NATARAJAN, P.; ZHU, J. **Fog computing: A platform for internet of things and analytics**. In: **BIG DATA AND INTERNET OF THINGS: A Roadmap for Smart Environments**. Cham: Springer, 2012. p. 169-186.

BORREGO, C. et al. **Assessment of air quality microsensors versus reference methods: The EuNetAir Joint Exercise – Part II**. Atmospheric environment (Oxford, England: 1994), v. 193, p. 127–142, 2018.

CELERY. Disponível em: https://docs.celeryproject.org/. Acesso em: 5 jul. 2024.

CHEN, M.; MAO, S.; LIU, Y. **Big Data: A survey**. Mobile Networks and Applications, v. 19, n. 2, p. 171-209, 2014.

DJANGO. Disponível em: https://www.djangoproject.com/. Acesso em: 5 jul. 2024.

DJANGO REST FRAMEWORK. Disponível em: https://www.django-rest-framework.org/. Acesso em: 5 jul. 2024.

DOCKER. Disponível em: https://www.docker.com/. Acesso em: 5 jul. 2024.

EUROPEAN ENVIRONMENT AGENCY. **Air quality in Europe - 2018 report**. 2018.

EVANS, E. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Addison-Wesley, 2003.

GOOGLE BIGQUERY. Disponível em: https://cloud.google.com/bigquery. Acesso em: 5 jul. 2024.

GOOGLE CLOUD. Disponível em: https://cloud.google.com/. Acesso em: 5 jul. 2024.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. **Internet of Things (IoT): A vision, architectural elements, and future directions**. Future Generation Computer Systems, v. 29, n. 7, p. 1645-1660, 2013.

HARRISON, R. M. et al. **Sources of ultrafine particles and chemical species along a traffic corridor: Comparison of road dust, particles generated by vehicle braking and tailpipe emissions**. Atmospheric Environment, v. 235, p. 117672, 2020.

HANES, D. et al. **IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things**. Cisco Press, 2017.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. **MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks**. In: **2008 3RD INTERNATIONAL CONFERENCE ON COMMUNICATION SYSTEMS SOFTWARE AND MIDDLEWARE AND WORKSHOPS**. [S.l.: s.n.], 2008. p. 791-798.

IEEE. 8275846 Abstract. Disponível em: https://ieeexplore.ieee.org/abstract/document/8275846. Acesso em: 5 jul. 2024.

INFLUXDATA. **InfluxDB Documentation**. 2023. Disponível em: https://docs.influxdata.com/influxdb/. Acesso em: 22 jun. 2024.

IQBAL, M. A. **Enabling the Internet of Things: Fundamentals, Design and Applications**. Springer, 2021.

JONES, M.; BRADLEY, J.; SAKIMURA, N. **JSON Web Token (JWT)**. RFC 7519, 2015.

KREPS, J.; NARKHEDE, N.; RAO, J. **Kafka: A Distributed Messaging System for Log Processing**. In: **PROCEEDINGS OF THE NETDB**. [S.l.: s.n.], 2011. v. 11, p. 1-7.

KUBERNETES. Disponível em: https://kubernetes.io/. Acesso em: 5 jul. 2024.

KUMAR, P. et al. **The rise of low-cost sensing for managing air pollution in cities**. Environment International, v. 75, p. 199-205, 2015.

KUNAK. **Kunak AIR Datasheet**. Disponível em: https://kunakair.com/doc/External/Kunak_AIR_Datasheet_EN.pdf. Acesso em: 5 jul. 2024.

LELIEVELD, J. et al. **Cardiovascular disease burden from ambient air pollution in Europe reassessed using novel hazard ratio functions**. European Heart Journal, v. 40, n. 20, p. 1590-1596, 2019.

MAKSIMOVIĆ, M. **The Role of Middleware in IoT Systems: Current Solutions and Research Challenges**. Wireless Personal Communications, v. 114, n. 2, p. 1-22, 2020. doi:10.1007/s11277-020-07483-9.

MARTIN, R. C. **Agile Software Development, Principles, Patterns, and Practices**. Prentice Hall, 2002.

MARTIN, R. C. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Prentice Hall, 2017.

MINISTÉRIO DO MEIO AMBIENTE. **Guia Técnico Qualidade do Ar**. Disponível em: https://antigo.mma.gov.br/images/agenda_ambiental/qualidade-do-ar/MMA-Guia-Tecnico-Qualidade-do-Ar.pdf. Acesso em: 5 jul. 2024.

MINISTÉRIO DO MEIO AMBIENTE. **Monitorar**. Disponível em: https://monitorar.mma.gov.br/. Acesso em: 5 jul. 2024.

MONGODB. Disponível em: https://www.mongodb.com/. Acesso em: 5 jul. 2024.

PALPANAS, T. **Big Data Analytics in the Dynamic World**. In: **PROCEEDINGS OF THE EDBT/ICDT WORKSHOPS**. [S.l.: s.n.], 2015. p. 3-6.

POPE, C. A.; EZZATI, M.; DOCKERY, D. W. **Fine-particulate air pollution and life expectancy in the United States**. New England Journal of Medicine, v. 360, n. 4, p. 376-386, 2009.

POSTGRESQL. **PostgreSQL Official Site**. 2023. Disponível em: https://www.postgresql.org/. Acesso em: 22 jun. 2024.

POWER BI. Disponível em: https://powerbi.microsoft.com/. Acesso em: 5 jul. 2024.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. **Low Power Wide Area Networks: An Overview**. IEEE Communications Surveys & Tutorials, v. 19, n. 2, p. 855-873, 2017.

REACT. Disponível em: https://reactjs.org/. Acesso em: 5 jul. 2024.

REDIS. Disponível em: https://redis.io/. Acesso em: 5 jul. 2024.

ROMAN, R.; ZHOU, J.; LOPEZ, J. **On the features and challenges of security and privacy in distributed Internet of Things**. Computer Networks, v. 57, n. 10, p. 2266-2279, 2013.

SADEGHI, A. R.; WACHSMANN, C.; WAIDNER, M. **Security and privacy challenges in industrial internet of things**. In: **PROCEEDINGS OF THE 52ND ANNUAL DESIGN AUTOMATION CONFERENCE**. [S.l.: s.n.], 2015. p. 1-6.

SEINFELD, J. H.; PANDIS, S. N. **Atmospheric Chemistry and Physics: From Air Pollution to Climate Change**. John Wiley & Sons, 2016.

SHELBY, Z.; HARTKE, K.; BORMANN, C.; FRANK, B. **Constrained Application Protocol (CoAP)**. RFC 7252, 2014.

SICARI, S.; RIZZARDI, A.; GRIECO, L. A.; COEN-PORISINI, A. **Security, privacy and trust in Internet of Things: The road ahead**. Computer Networks, v. 76, p. 146-164, 2015.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. **Edge Computing: Vision and Challenges**. IEEE Internet of Things Journal, v. 3, n. 5, p. 637-646, 2016. doi:10.1109/JIOT.2016.2579198.

SNYDER, E. G. et al. **The changing paradigm of air pollution monitoring**. Environmental Science & Technology, v. 47, n. 20, p. 11369-11377, 2013.

STACK OVERFLOW DEVELOPER SURVEY 2023. Disponível em: https://stackoverflow.com/survey/2023. Acesso em: 5 jul. 2024.

SUO, H.; WAN, J.; ZOU, C.; LIU, J. **Security in the Internet of Things: A Review**. In: **2012 INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND ELECTRONICS ENGINEERING**, v. 3, p. 648-651, 2012. IEEE.

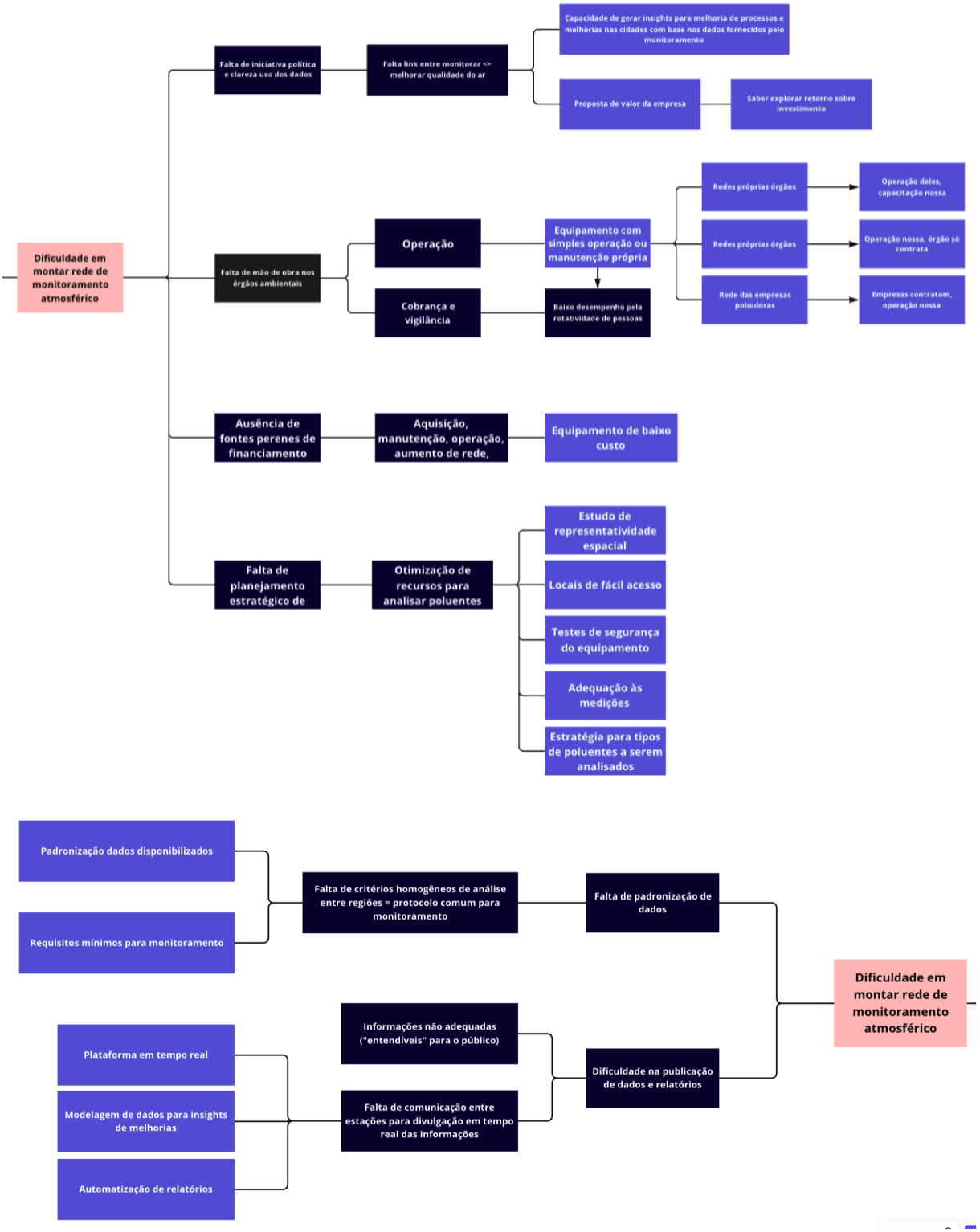
TIMESCALE. **TimescaleDB Documentation**. 2023. Disponível em: https://docs.timescale.com/. Acesso em: 22 jun. 2024.

WORLD HEALTH ORGANIZATION. **Ambient (outdoor) air pollution**. 2021.

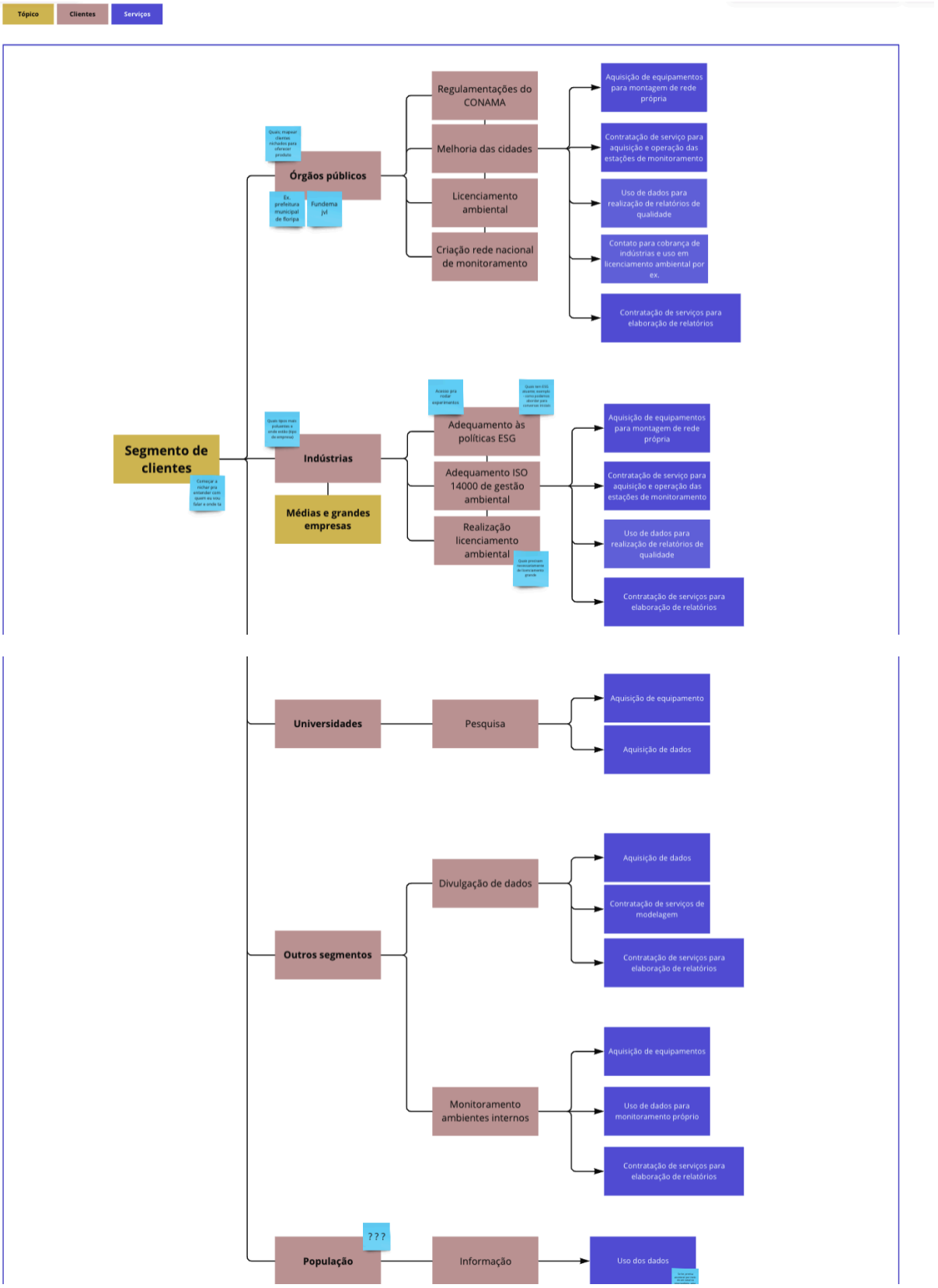
ZHANG, Y.; QIU, M.; TSAI, C. W.; HASSAN, M. M.; ALAMRI, A. **Health-CPS: Healthcare Cyber-Physical System Assisted by Cloud and Big Data**. IEEE Systems Journal, v. 11, n. 1, p. 88-95, 2015.

ZIGBEE ALLIANCE. **ZigBee Specification**. 2012.

APÊNDICE A – WHITEBOARD PROPOSTA DE VALOR



APÊNDICE B – WHITEBOARD SEGMENTAÇÃO POR STAKEHOLDER



APÊNDICE C – Motivos Para Escolha do Web Framework

O framework escolhido para o desenvolvimento do backend foi o Django. O motivo por trás da escolha se deu principalmente pelos critérios abaixo:

1. **Facilidade de Contratação de Desenvolvedores:**

Django é um framework web popular e amplamente adotado na comunidade de desenvolvedores Python. Em uma pesquisa realizada pelo Stack Overflow em 2023, Python foi listado como uma das linguagens de programação mais populares e Django como um dos frameworks mais usados. Isso facilita a contratação de desenvolvedores qualificados que já possuem experiência com Django, reduzindo o tempo de onboarding e aumentando a produtividade da equipe.

2. **Tamanho da Comunidade:**

A comunidade de Django é uma das maiores e mais ativas no mundo Python. Segundo o Python Developers Survey de 2022, Django foi um dos frameworks mais mencionados pelos desenvolvedores, refletindo uma comunidade grande e ativa. Isso significa que há uma abundância de recursos, como bibliotecas, documentação detalhada, e fóruns de discussão, onde desenvolvedores podem encontrar respostas para problemas comuns e compartilhar conhecimentos.

3. **Ferramentas Úteis:**

Django vem com um conjunto robusto de ferramentas integradas que facilitam o desenvolvimento de aplicações web seguras e escaláveis. Por exemplo, o Django Admin é uma interface de administração que permite aos desenvolvedores gerenciar dados da aplicação de maneira fácil e eficiente.

4. **Django Admin resultaria em uma interface pronta para a gestão dos equipamentos out-of-the-box:**

O Django Admin resultou em uma interface pronta para a gestão dos equipamentos "out-of-the-box". Isso significa que, com configuração mínima, foi utilizado o Django Admin para criar, ler, atualizar e deletar (CRUD) registros de equipamentos e outros dados relacionados. A interface do Django Admin é altamente personalizável e pode ser adaptada

para atender às necessidades específicas do projeto, atendendo as demandas da V1 sem tempo agregado no desenvolvimento de um Frontend.

APÊNDICE D – COMPARAÇÃO COM INFLUXDB

Os bancos de dados foram comparados nos seguintes critérios:

1. **Integração e Facilidade de Uso:**

TimescaleDB: Beneficia-se da ampla adoção e familiaridade do PostgreSQL, facilitando a integração com ferramentas e sistemas existentes e reduzindo a curva de aprendizado para desenvolvedores que já conhecem SQL.

InfluxDB: Oferece uma experiência altamente otimizada para dados temporais, mas demandaria a implementação de regras de negócio adicionais para integração com o banco de dados PostgreSQL usado para o gerenciamento dos equipamentos.

2. **Performance e Escalabilidade:**

TimescaleDB: Escalável e robusto para grandes volumes de dados temporais, com benefícios adicionais de compressão e particionamento automáticos. Em benchmarks realizados pela Timescale, foi demonstrado que TimescaleDB pode processar até 10 bilhões de linhas de dados temporais com consultas complexas executadas em segundos (Timescale, 2023).

InfluxDB: Excelência em ingestão rápida de dados e consultas temporais, sendo ideal para aplicações de monitoramento intensivo de dados. InfluxDB consegue inserir até 800.000 pontos por segundo em hardware apropriado (InfluxData, 2023).

3. **Funcionalidades Avançadas:**

TimescaleDB: Suporta funcionalidades avançadas do PostgreSQL, como joins complexos, transações ACID, e integrações com diversas ferramentas de análise e visualização.

InfluxDB: Oferece funcionalidades específicas de séries temporais, como downsampling e políticas de retenção, mas pode ser limitado em casos que exigem operações relacionais complexas.

APÊNDICE E – Escopo Priorizado

Em verde, estão elencados os requisitos funcionais priorizados para o escopo deste trabalho.

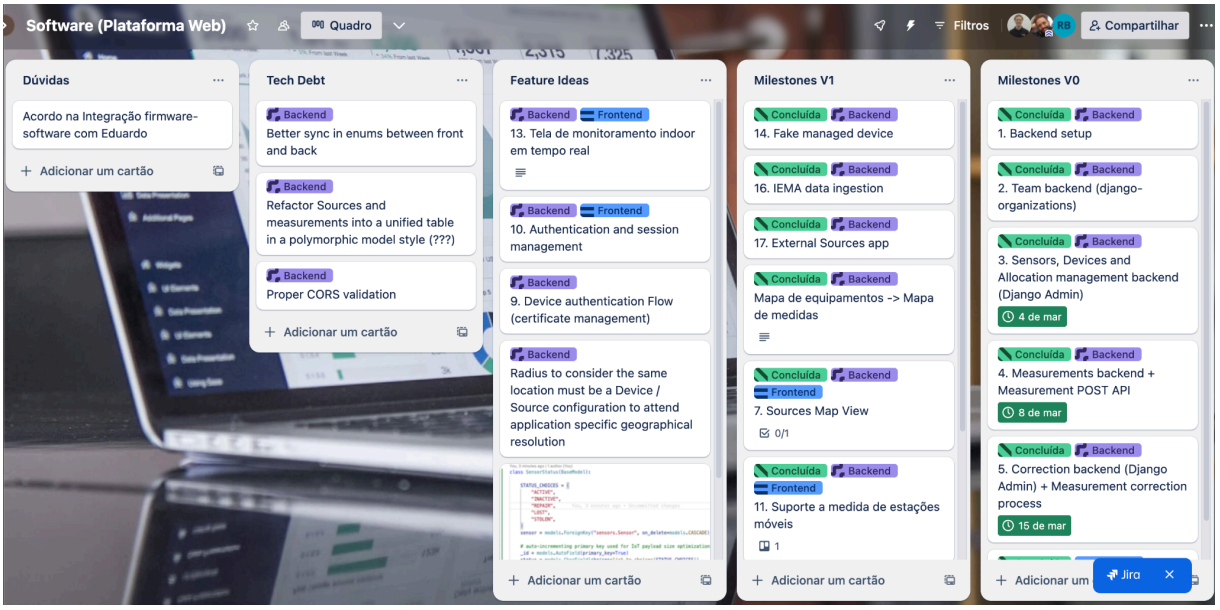
Tabela 3 - Listagem dos requisitos funcionais priorizados

Categoria / Requisito		Importância
Conectividade e Coleta dos dados		
	O equipamento deve ser capaz de enviar dados em tempo real ao servidor	Must have
	O sistema deve receber dados de sensores com períodos de amostragem diferentes	Must have
	O sistema deve ter um mecanismo de redundância para envio de dados em caso de falha na conexão.	Nice to have
Acessos e Gerenciamento de Usuários		
	O sistema deve permitir o registro e gerenciamento de clientes	Must have
	Cada cliente deve ter acesso a um ambiente exclusivo	Must have
	Usuários podem acessar o ambiente de mais de um cliente	Nice to have
	Cada usuário deve ter um papel dentro do time (e.g.: admin/analista/operador/externo/...) com diferentes permissões	Nice to have
Gerenciamento da Frota		
	É necessário manter um registro dos equipamentos montados	Nice to have
	É necessário manter um registro dos sensores adquiridos e em quais equipamentos os sensores foram instalados	Nice to have
	Um equipamento pode ser alugado ou vendido para um cliente. Isso será chamado de alocação	Nice to have
	O sistema deve gerar alertas quando um sensor ou equipamento necessita de manutenção.	Nice to have
	O sistema deve permitir a fácil atualização de firmware dos sensores remotamente.	Nice to have
	O sistema deve ser capaz de operar equipamentos de terceiros, por exemplo, estações de referência	Nice to have
Flexibilidade do equipamento		
	O equipamento comporta diferentes configurações de sensores	Must have
Processamento das medidas		
	Cada sensor tem constantes definidas em datasheet que serão utilizadas para o processamento dos dados	Must have
	Os dados brutos gerados por cada sensor devem ser processados de acordo com as recomendações do fabricante para o sensor	Must have
	O sistema deve manter rastreabilidade dos dados processados	Must have
	O sistema deve permitir a aplicação de algoritmos de calibração para melhorar a precisão dos dados dos sensores.	Must have
Integração de bases de dados		
	O sistema deve ser capaz de se conectar a sistemas de monitoramento já existentes (públicos ou privados)	Must have

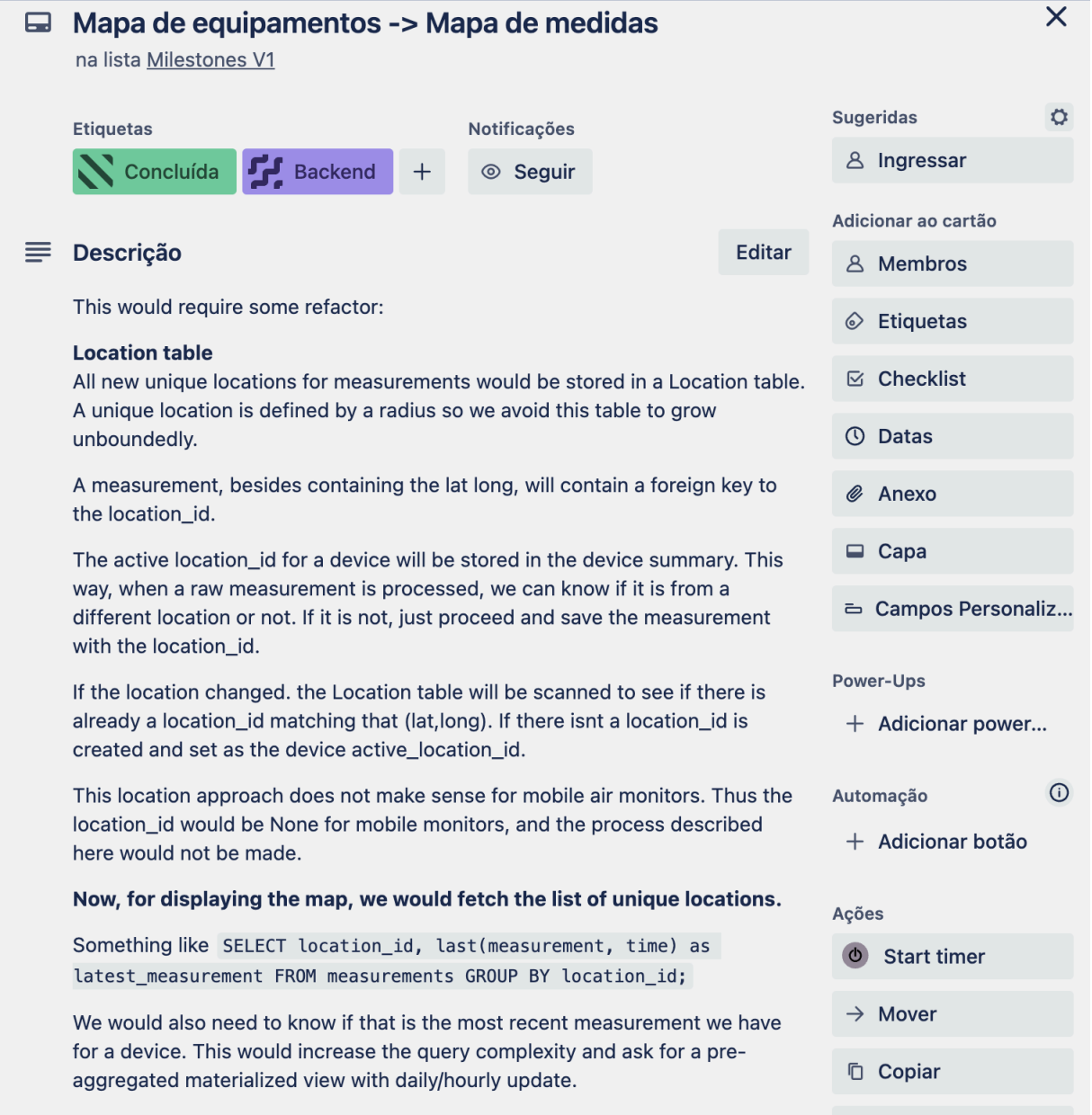
	O sistema deve poder consumir/consolidar dados de outras plataformas	Must have
	O sistema deve poder fornecer dados para outras plataformas	Must have
	O sistema deve permitir a exportação dos dados em diferentes formatos (e.g., CSV, JSON, XML) para análise externa.	Must have
Interoperabilidade		
	O sistema deve ser compatível com normas e padrões nacionais (CONAMA) de monitoramento ambiental.	Must have
Notificações		
	Permitir a configuração dinâmica de limites em uma granularidade temporal para cada variável do sistema	Nice to have
	O sistema deve permitir a configuração de diferentes níveis de severidade para as notificações.	Nice to have
Dashboard/Análise dos dados		
	Analisar dados de estações de baixo custo e estações de referência	Must have
	Visualização espacial das estações resumidas	Must have
	Visualização detalhada das estações	Must have
	O sistema deve mostrar as concentrações ao longo do tempo do tempo	Must have
	Capacidade de comparar as medições entre estações de referência e baixo custo co-locadas para fins de validação da calibração	Nice to have
Visibilidade dos Dados		
	Somente o cliente tem acesso aos dados gerados pelo equipamento no período em que o equipamento estava alocado, exceto que o cliente decida publicas esses dados.	Must have
	Um cliente pode habilitar uma página pública na qual qualquer usuário, mesmo não autenticado teria acesso (por exemplo, um orgão público disponibilizando os dados para a sociedade civil)	Nice to have
	Os dados disponíveis na página pública do cliente serão habilitados por alocação de equipamento.	Nice to have
Gerenciamento das Calibrações		
	A calibração de um equipamento é atrelada aos sensores instalados no equipamento;	Must have
	Um equipamento pode ser recalibrado;	Nice to have
	Orquestrar o treinamento de modelos de calibração na nuvem	Nice to have
Integração de diferentes tipos de fontes		
	Satelite	Nice to have
	Inventário de emissões	Nice to have
	Modelagens	Nice to have
Monitoramento de ambientes Internos		Nice to have

APÊNDICE F – ESTRUTURAÇÃO DAS TAREFAS E GERENCIAMENTO DO PROJETO

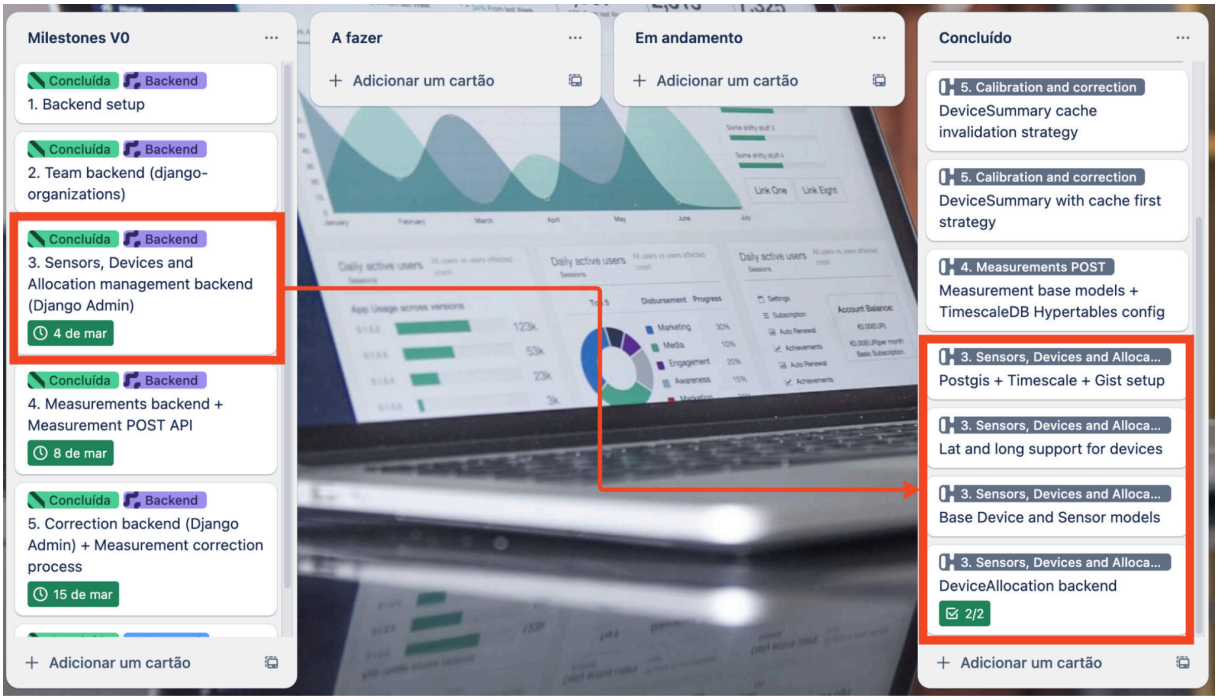
Para a gestão do projeto foi utilizada a ferramenta Trello. Na figura abaixo podemos entender como foi feita a divisão. Importante destacar que ao longo do desenvolvimento do projeto, alguns candidatos à refatoração foram criados e documentados na lista "Tech Debt". Esses pontos serão detalhados na seção final do desenvolvimento.



Na figura abaixo, é exemplificada a descrição de uma Epic. A Epic em questão não foi contemplada na arquitetura inicial, mas identificada ao longo do desenvolvimento da Milestone V0. Esta foi então adicionada à lista Tech Debt e priorizada para ser desenvolvida na Milestone V1.



Na figura abaixo podemos entender como uma Epic divide-se em uma ou mais Tasks. As Tasks seguem um padrão de documentação e detalhamento similar às Epics.


Milestones V0
Concluída
Backend
1. Backend setup
Concluída
Backend
2. Team backend (django-organizations)
Concluída
Backend
3. Sensors, Devices and Allocation management backend (Django Admin)
4 de mar
Concluída
Backend
4. Measurements backend + Measurement POST API
8 de mar
Concluída
Backend
5. Correction backend (Django Admin) + Measurement correction process
15 de mar
Adicionar um cartão
A fazer
Adicionar um cartão
Em andamento
Adicionar um cartão
Concluído
5. Calibration and correction
DeviceSummary cache invalidation strategy
5. Calibration and correction
DeviceSummary with cache first strategy
4. Measurements POST
Measurement base models + TimescaleDB Hypertables config
3. Sensors, Devices and Alloca...
Postgis + Timescale + Gist setup
3. Sensors, Devices and Alloca...
Lat and long support for devices
3. Sensors, Devices and Alloca...
Base Device and Sensor models
3. Sensors, Devices and Alloca...
DeviceAllocation backend
2/2
Adicionar um cartão

APÊNDICE G – Implementação da Classe SensorDataProcessor

models.py | sensor_data_processors.py ×

iris_backend > managed_devices > sensors > services > sensor_data_processors.py > ...

```python
processors: dict[str, type["SensorDataProcessor"]] = {
    # modelId: SensorDataProcessor
}


You, 3 months ago | 1 author (You)
@dataclass
class SensorProcessedVariable(ProcessedVariable):
    """Adds the sensor_ids_chain to the ProcessedVariable.

    This id chain is used to track the sensors that were used to process this variable.
    """

    sensor_ids_chain: list[int]


def _register_processor(processor: type["SensorDataProcessor"]):
    for model in processor._MODELS:  # type: ignore
        if model in processors:
            raise ValueError(f"Processor for model {model} already registered")

        processors[model] = processor


def get_sensor_data_processor(model: str, sensor_summary: SensorSummary) -> "SensorDataProcessor":
    try:
        return processors[model](sensor_summary)
    except KeyError:
        raise ValueError(f"No processor registered for model {model}")

    You, 3 months ago • save measurements view + device data processor …
You, 3 months ago | 1 author (You)
class SensorDataProcessor:

    _MODELS: Iterable[SensorsModelsLiteral] = NotImplemented

    ExpectedSensorReading: type[pydantic.BaseModel] = NotImplementedType
    ExpectedComplementaryVariables: list[MeasurementVariablesEnum] = NotImplemented

    def __init_subclass__(cls) -> None:
        super().__init_subclass__()
        if cls._MODELS is not NotImplemented:
            _register_processor(cls)

    def __init__(self, sensor_summary: SensorSummary) -> None:
        self.sensor_summary = sensor_summary

    def process(
        self,
        raw_sensor_reading: dict | pydantic.BaseModel,
        complementary_variables: None | list[SensorProcessedVariable] = None,
    ) -> list[SensorProcessedVariable]:
        if isinstance(raw_sensor_reading, dict):
            try:
                sensor_reading = self.ExpectedSensorReading(**raw_sensor_reading)
            except pydantic.ValidationError as e:
                raise ValidationError(e)
        elif isinstance(raw_sensor_reading, self.ExpectedSensorReading):
            sensor_reading = raw_sensor_reading
        else:
            raise ValueError(f"Invalid sensor reading type: {type(raw_sensor_reading)}")

        processed_variables = self._process(sensor_reading, complementary_variables)

        sensor_ids_chain = []
        if complementary_variables:
            for complementary_variable in complementary_variables:
                sensor_ids_chain.extend(complementary_variable.sensor_ids_chain)

        sensor_ids_chain.append(self.sensor_summary.id)

        return [
            SensorProcessedVariable(
                value=processed_variable.value, variable=processed_variable.variable, sensor_ids_chain=sensor_ids_chain
            )
```

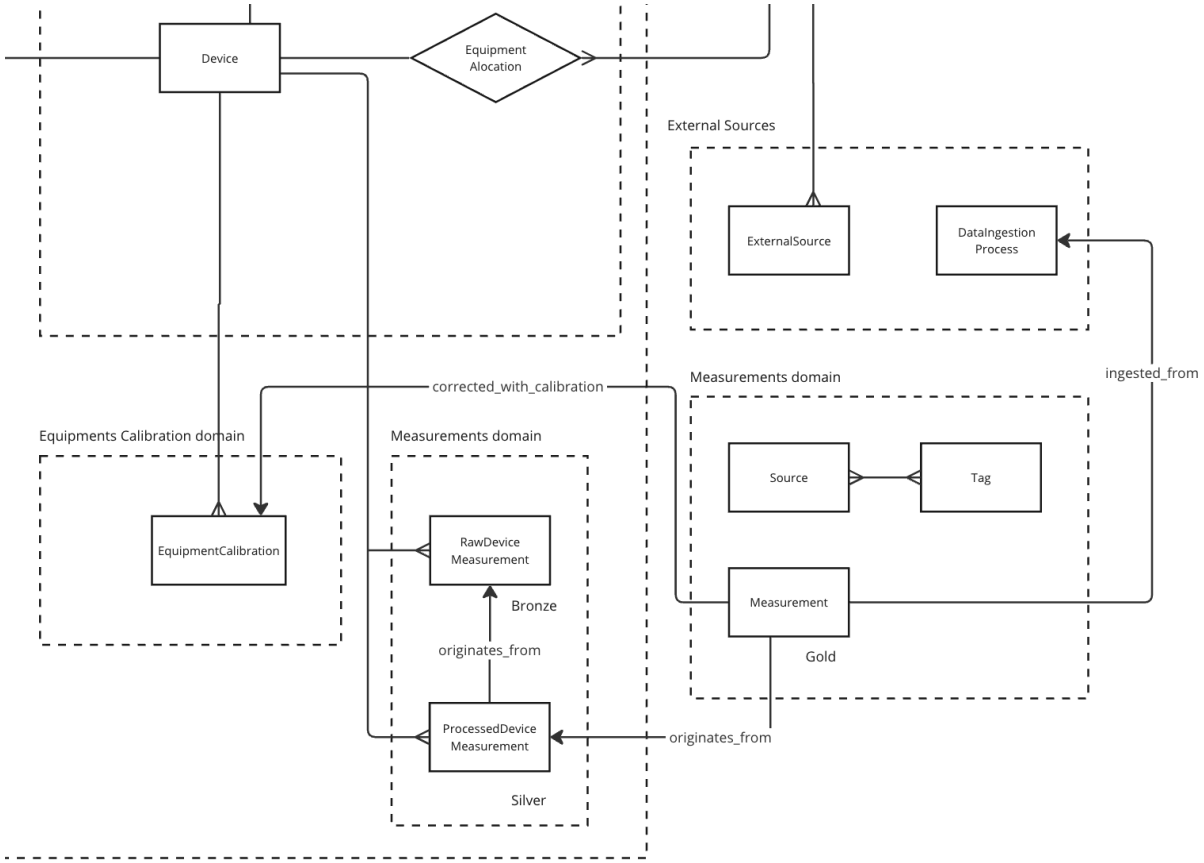
(parameter) model: str

APÊNDICE H - Futuras Refatorações E Aprendizados Da Arquitetura Inicial

Uma outra forma de modelar o problema da consolidação entre Managed Devices e External Sources:

A diferença esperada entre os dois era maior no início do projeto, no entanto esta se deu apenas na forma como as medidas são originadas. Todas as regras de negócio de consumo dessas informações são muito similares. Desta forma, uma modelagem mais eficiente seria criar um domínio **Measurements** responsável por: generalizar **Device**s e **ExternalSource**s por uma interface **Source.** E centralizar todas as medidas Gold na entidade **Measurement** permitindo um esquema flexível para o armazenamento de metadados da originação**.** Esta flexibilização do esquema não traria um custo na análise dos dados, uma vez que estas são informações existentes apenas para fins de auditoria e traceability dos dados e não são acessadas ativamente na aplicação.

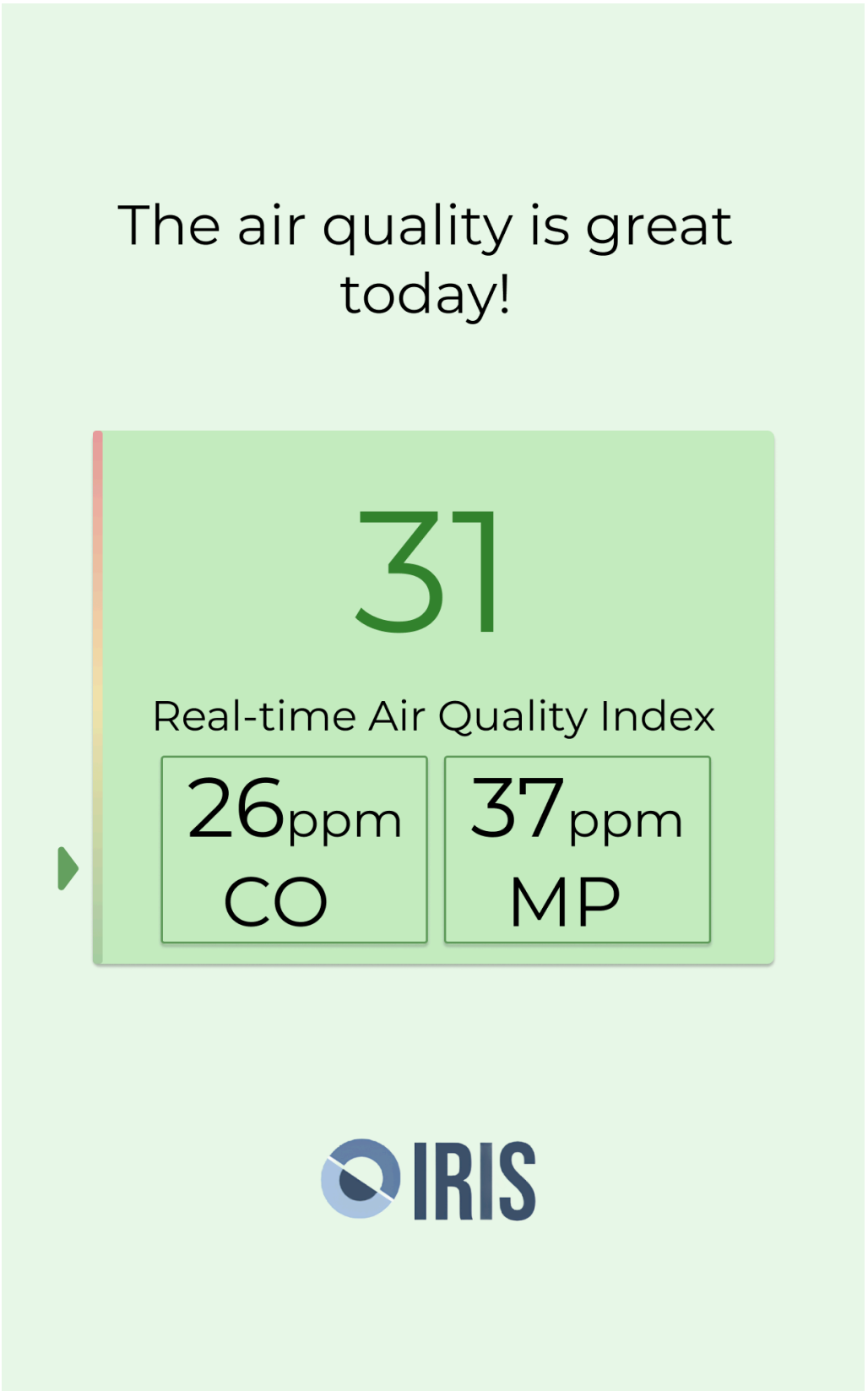
Um **Source** teria atributos para identificar o tipo da source e a origem dos dados. Esta modelagem diminui o retrabalho implementado nos dois domínios, ao custo de um pouco mais de acoplamento, no entanto seria a melhor opção. Importante destacar que esse não é um *refactor* difícil de ser implementado dado às boas práticas na implementação dos outros domínios. O tempo estimado para essa refatoração foi de ~12h e foi despriorizado pelo baixo impacto frente a alocação do autor em uma frente comercial que pudesse garantir o primeiro cliente.



Sincronização de enumeradores entre os serviços

Uma forma de evitar duplicação de enums seria a disponibilização destes via API. Na implementação atual, por conta da redundância, esses enums não coincidem. Desta forma são necessários alguns workarounds na API e no Front para que a integração funcione.

APÊNDICE I – WIREFRAMES DO SISTEMA



A qualidade do ar neste ambiente está regular!

0,42

indíce da qualidade do ar

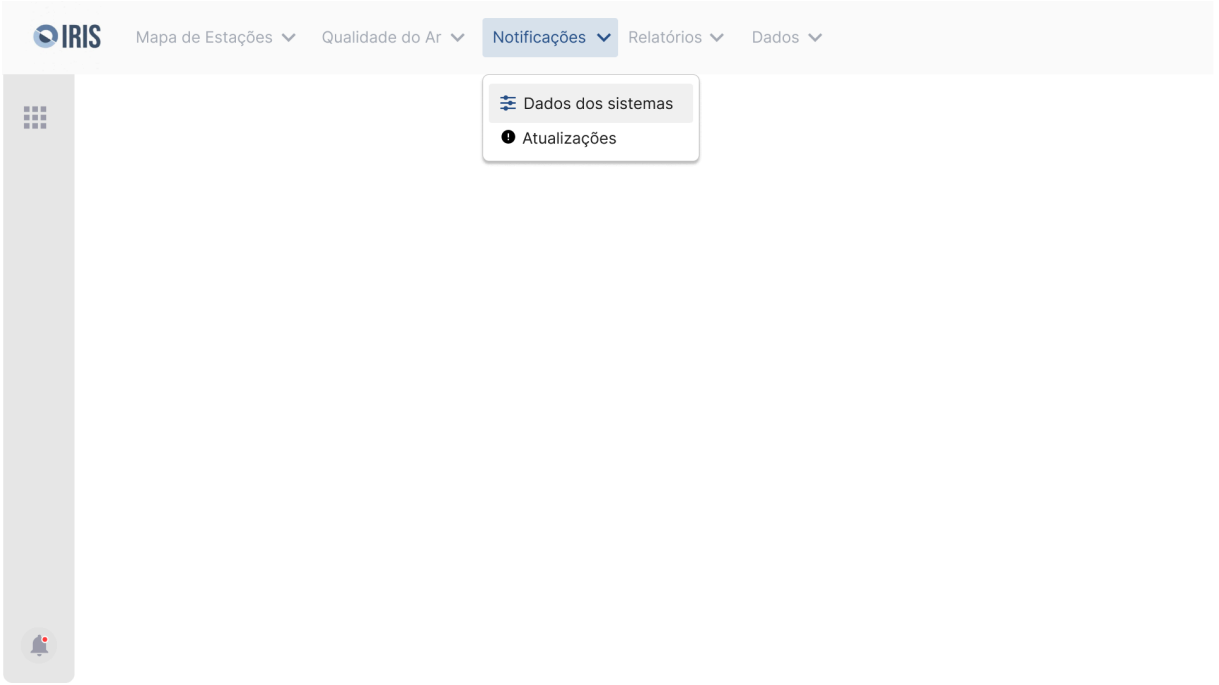
52ppm CO2

50ppm MP

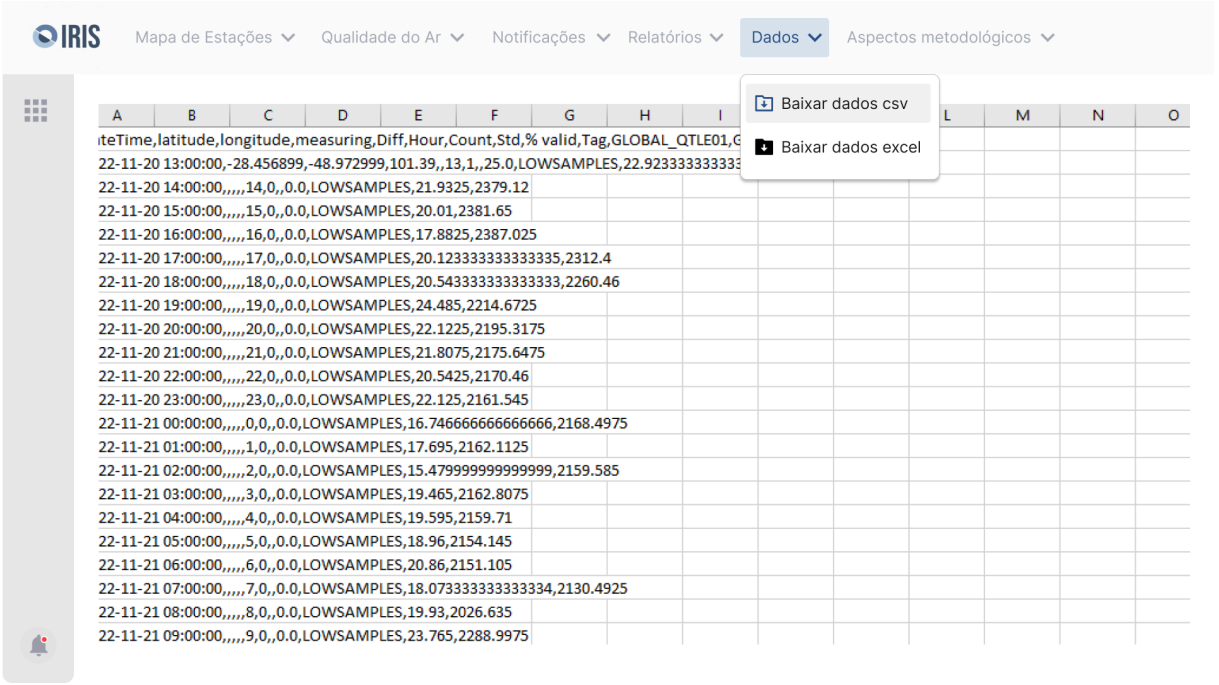
Startup Summit 2024 | IRIS


Área do cliente
Login
Senha
Esqueci minha senha
Acessar
IRIS
new environmental softwares and systems

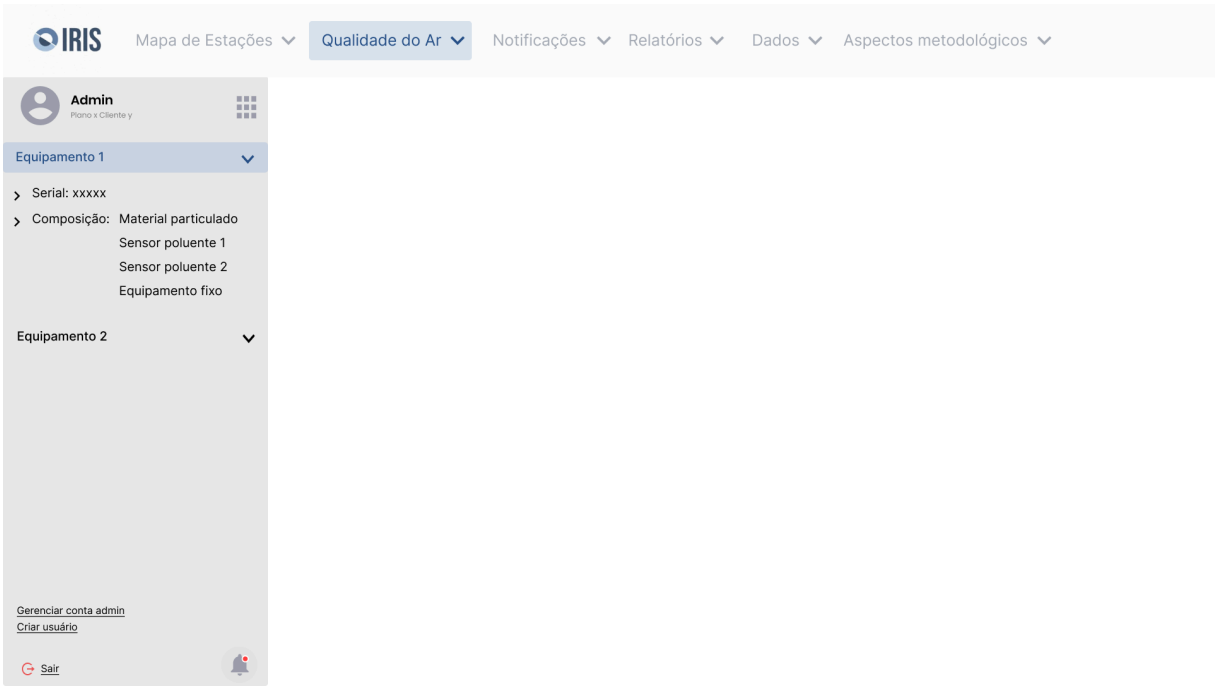

IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Minhas estações
Todas as estações visíveis
Venezuela
Guiana
Colômbia
Equador
Peru
Brazil
Bolívia
Chile
Paraguai
© 2023 Mapbox © OpenStreetMap
Poluente
(Tudo)
Órgão Gestor
SEMAD - GO
SEMACE - CE
Prefeitura do Rio de Janeiro
INEMA - BA
INEA - RJ
IEMA - ES
IBRAM - DF
IAT - PR
FEPAM - RS
FEAM - MG
CPRH - PE
CETESB - SP
Ano
2021
Estação: Osasco
Estado - Município: SP - Osasco
Poluente monitorados: CO, MP2.5, MP10, NO2, PTS, SO2
Órgão gestor: CETESB - SP
Escala espacial: Micro
Tipo de monitoramento: Automática, Manual
Histórico dos dados: 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022

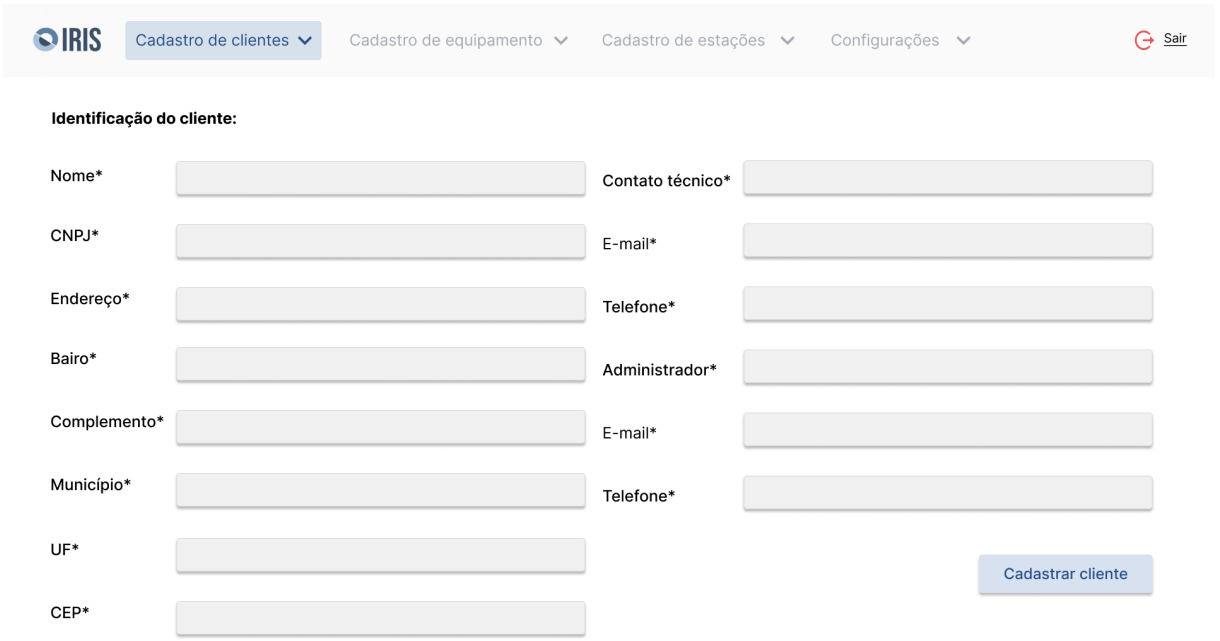

IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Dados dos sistemas
Atualizações

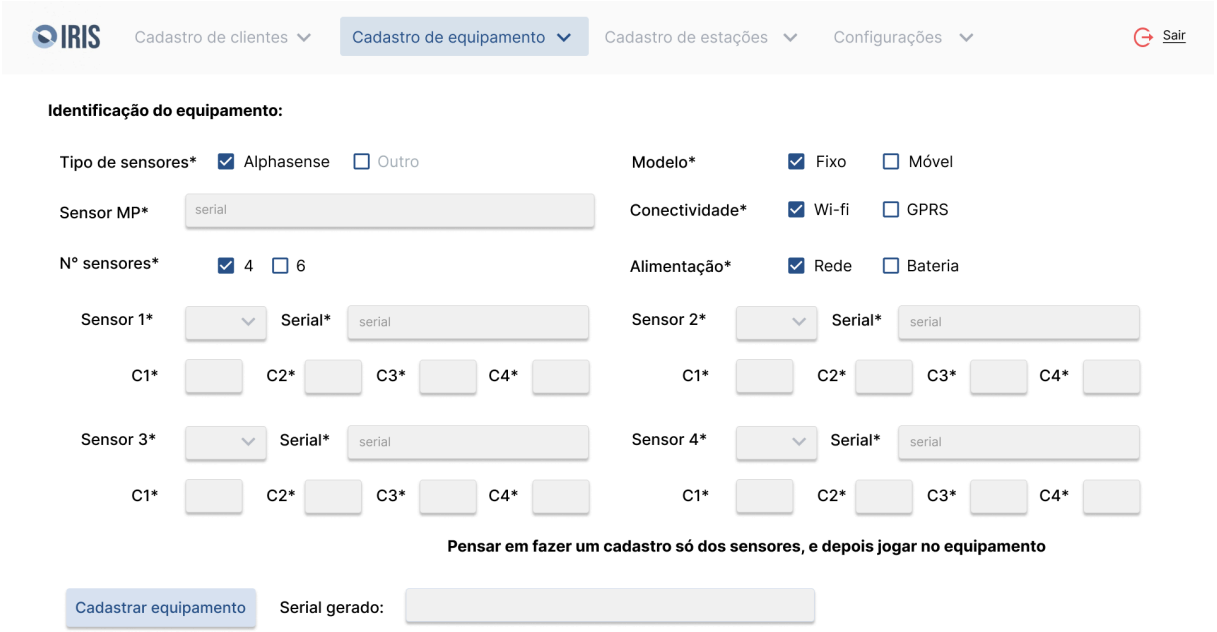

IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Gerar Relatório
RELATÓRIO DA QUALIDADE DO AR
Primeiro semestre de 2023
IRIS new environmental softwares and systems

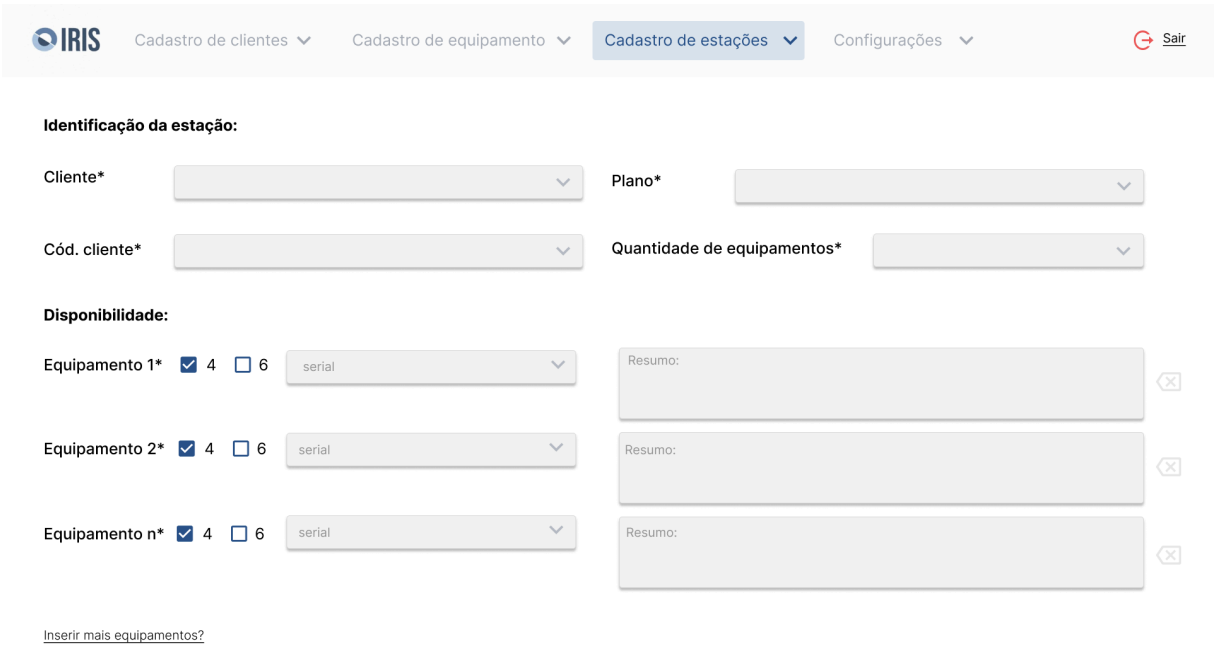

IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Aspectos metodológicos
Baixar dados csv
Baixar dados excel
iteTime,latitude,longitude,measuring,Diff,Hour,Count,Std,% valid,Tag,GLOBAL_QTLE01,G
22-11-20 13:00:00,-28.456899,-48.972999,101.39,,13,1,,25.0,LOWSAMPLES,22.923333333333
22-11-20 14:00:00,,,,,14,0,,0.0,LOWSAMPLES,21.9325,2379.12
22-11-20 15:00:00,,,,,15,0,,0.0,LOWSAMPLES,20.01,2381.65
22-11-20 16:00:00,,,,,16,0,,0.0,LOWSAMPLES,17.8825,2387.025
22-11-20 17:00:00,,,,,17,0,,0.0,LOWSAMPLES,20.123333333333335,2312.4
22-11-20 18:00:00,,,,,18,0,,0.0,LOWSAMPLES,20.543333333333333,2260.46
22-11-20 19:00:00,,,,,19,0,,0.0,LOWSAMPLES,24.485,2214.6725
22-11-20 20:00:00,,,,,20,0,,0.0,LOWSAMPLES,22.1225,2195.3175
22-11-20 21:00:00,,,,,21,0,,0.0,LOWSAMPLES,21.8075,2175.6475
22-11-20 22:00:00,,,,,22,0,,0.0,LOWSAMPLES,20.5425,2170.46
22-11-20 23:00:00,,,,,23,0,,0.0,LOWSAMPLES,22.125,2161.545
22-11-21 00:00:00,,,,,0,0,,0.0,LOWSAMPLES,16.746666666666666,2168.4975
22-11-21 01:00:00,,,,,1,0,,0.0,LOWSAMPLES,17.695,2162.1125
22-11-21 02:00:00,,,,,2,0,,0.0,LOWSAMPLES,15.479999999999999,2159.585
22-11-21 03:00:00,,,,,3,0,,0.0,LOWSAMPLES,19.465,2162.8075
22-11-21 04:00:00,,,,,4,0,,0.0,LOWSAMPLES,19.595,2159.71
22-11-21 05:00:00,,,,,5,0,,0.0,LOWSAMPLES,18.96,2154.145
22-11-21 06:00:00,,,,,6,0,,0.0,LOWSAMPLES,20.86,2151.105
22-11-21 07:00:00,,,,,7,0,,0.0,LOWSAMPLES,18.073333333333334,2130.4925
22-11-21 08:00:00,,,,,8,0,,0.0,LOWSAMPLES,19.93,2026.635
22-11-21 09:00:00,,,,,9,0,,0.0,LOWSAMPLES,23.765,2288.9975


IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Aspectos metodológicos
Padrões da qualidade do ar no Brasil
Índice da Qualidade do Ar (IQAr)
Recomendações e métricas
O Decreto Estadual nº 59113/2013 estabelece que a administração da qualidade do ar no território do Estado de São Paulo será efetuada através de Padrões de Qualidade do Ar, observados os seguintes critérios:
1. Metas Intermediárias – (MI) estabelecidas como valores temporários a serem cumpridos em etapas, visando à melhoria gradativa da qualidade do ar no Estado de São Paulo, baseada na busca pela redução das emissões de fontes fixas e móveis, em linha com os princípios do desenvolvimento sustentável;
2. Padrões Finais (PF) – Padrões determinados pelo melhor conhecimento científico considerando as menores concentrações possíveis no contexto de limitações locais, capacidade técnica e prioridades em termos de saúde pública para que a saúde da população seja preservada ao máximo em relação aos danos causados pela poluição atmosférica.
As Metas Intermediárias devem ser atendidas em 3 (três) etapas:
1. Meta Intermediária Etapa 1 – (MI1) – Valores de concentração de poluentes atmosféricos a serem respeitados a partir de 24/04/2013. Estes valores ficaram vigentes até 31/12/2021.
2. Meta Intermediária Etapa 2 – (MI2) – Valores de concentração de poluentes atmosféricos que devem ser respeitados subsequentemente à MI1, que entrará em vigor após avaliações realizadas na Etapa 1, reveladas por estudos técnicos apresentados pelo órgão ambiental estadual, convalidados pelo CONSEMA. a MI2 entrou em vigor a partir de 01/01/2022 (Deliberação CONSEMA nº 4, de 19/05/2021, publicada no DOE de 26/05/2021).
3. Meta Intermediária Etapa 3 – (MI3) – Valores de concentração de poluentes atmosféricos que devem ser respeitados nos anos subsequentes à MI2, sendo que seu prazo de duração será definido pelo CONSEMA, a partir do início da sua vigência, com base nas avaliações realizadas na Etapa 2.
Padrões Estaduais de Qual
(Decreto Estadual nº 5911
Poluente
Tempo
Amostragem
partículas inaláveis (MP_{10})
24 horas
MAA¹
120
40
100
35
75
30
50
20
partículas inaláveis finas ($MP_{2,5}$)
24 horas
MAA¹
60
20
50
17
37
15
25
10
dióxido de enxofre (SO_2)
24 horas
MAA¹
60
40
40
30
30
20
20
–
dióxido de nitrogênio (NO_2)
1 hora
MAA¹
260
60
240
50
220
45
200
40
Ozônio (O_3)
8 horas
140
130
120
100
monóxido de carbono (CO)
8 horas
–
–
–
9 ppm


IRIS
Mapa de Estações
Qualidade do Ar
Notificações
Relatórios
Dados
Aspectos metodológicos
Admin
Equipamento 1
Serial: xxxxx
Composição: Material particulado
Sensor poluente 1
Sensor poluente 2
Equipamento fixo
Equipamento 2
Gerenciar conta admin
Criar usuário
Sair


IRIS
Cadastro de clientes
Cadastro de equipamento
Cadastro de estações
Configurações
Sair
Identificação do cliente:
Nome*
CNPJ*
Endereço*
Bairro*
Complemento*
Município*
UF*
CEP*
Contato técnico*
E-mail*
Telefone*
Administrador*
E-mail*
Telefone*
Cadastrar cliente


IRIS
Cadastro de clientes
Cadastro de equipamento
Cadastro de estações
Configurações
Sair
Identificação do equipamento:
Tipo de sensores* Alphasense Outro
Modelo* Fixo Móvel
Sensor MP* serial
Conectividade* Wi-fi GPRS
N° sensores* 4 6
Alimentação* Rede Bateria
Sensor 1* Serial* serial
Sensor 2* Serial* serial
C1* C2* C3* C4*
C1* C2* C3* C4*
Sensor 3* Serial* serial
Sensor 4* Serial* serial
C1* C2* C3* C4*
C1* C2* C3* C4*
Pensar em fazer um cadastro só dos sensores, e depois jogar no equipamento
Cadastrar equipamento
Serial gerado:


IRIS
Cadastro de clientes
Cadastro de equipamento
Cadastro de estações
Configurações
Sair
Identificação da estação:
Cliente*
Plano*
Cód. cliente*
Quantidade de equipamentos*
Disponibilidade:
Equipamento 1* 4 6 serial
Resumo:
Equipamento 2* 4 6 serial
Resumo:
Equipamento n* 4 6 serial
Resumo:
Inserir mais equipamentos?