Universidade Federal de Goiás
Instituto de Informática
Padrões de Arquitetura de Software

MediAr

Documento de Arquitetura de Software

André Lopes, 201703739
Rogério Rodrigues Rocha, 202203530

Goiânia
Outubro de 2024

1. Introdução

1.1 Finalidade

O objetivo deste documento é descrever a arquitetura do sistema de monitoramento de poluição urbana MediAr. Ele é destinado às partes interessadas, incluindo desenvolvedores, gerentes de projeto e a equipe de implantação. O foco deste documento é garantir que a arquitetura atenda aos requisitos funcionais e não funcionais, proporcionando uma base sólida para o desenvolvimento e a manutenção do sistema.

1.2 Escopo

Este documento abrange os requisitos funcionais e não funcionais, detalhando os aspectos arquiteturais, os padrões utilizados, e as visões da arquitetura que serão implementadas no sistema. O sistema permitirá o monitoramento da qualidade do ar em tempo real, consultas históricas e comparações de dados de poluição, bem como informações educativas sobre os impactos ambientais.

1.3 Definições, Acrônimos e Abreviações

Aplicativo: Software destinado ao monitoramento de poluição.

Poluição Urbana: Qualidade do ar medida em áreas urbanas.

Stakeholder: Indivíduo ou grupo interessado no sistema (usuários, desenvolvedores, gerentes de projeto).

Visão Arquitetural: Representação de aspectos da arquitetura do sistema para um público específico.

Ponto de Vista Arquitetural: Abordagem utilizada para descrever e interpretar uma visão arquitetural.

Cliente-Servidor: Modelo de arquitetura de software em que o cliente faz requisições ao servidor, que processa e retorna as respostas. O cliente é a interface com o usuário, enquanto o servidor gerencia a lógica de negócio e o armazenamento de dados.

REST: Estilo de arquitetura para comunicação entre sistemas distribuídos. Utiliza métodos HTTP (GET, POST, PUT, DELETE) e recursos identificados por URLs para troca de dados entre clientes e servidores de forma simples e escalável.

MVC (Model-View-Controller): Padrão de arquitetura que separa a aplicação em três componentes principais: Model (responsável pela lógica e dados), View (interface com o usuário) e Controller (responsável pela comunicação entre Model e View). Esse padrão facilita a organização e manutenção de sistemas.

Frontend (Cliente): Parte da aplicação que interage diretamente com o usuário, geralmente composta por interfaces gráficas desenvolvidas com tecnologias como HTML, CSS e JavaScript. O frontend é responsável por enviar requisições ao backend e exibir os resultados.

Backend (Servidor): Parte da aplicação responsável pelo processamento de dados, lógica de negócios e integração com bancos de dados e APIs. O backend lida com as requisições do frontend e retorna as respostas adequadas, sendo executado no servidor.

Banco de Dados: Sistema de armazenamento e gerenciamento de dados estruturados ou não estruturados. Ele é utilizado pela aplicação para armazenar informações de maneira persistente e fornecer dados ao backend conforme necessário.

API: Conjunto de definições e protocolos que permite a comunicação entre diferentes sistemas. As APIs facilitam a integração de funcionalidades entre o frontend e o backend, bem como entre diferentes serviços.

1.4 Referências

ISO/IEC/IEEE 42010 - Normas para Descrição Arquitetural.
ISO/IEC 9126 - Normas para Qualidade de Software.
4+1 View Model - Modelo de representação arquitetural.

1.5 Visão Geral

O documento detalha os requisitos e restrições, os atributos de qualidade priorizados (segurança, usabilidade, escalabilidade), e os padrões arquiteturais aplicados (Cliente-Servidor, REST e MVC). As visões arquiteturais são descritas para comunicar a estrutura do sistema.

2. Contexto da Arquitetura

2.1 Funcionalidades e Restrições Arquiteturais

O sistema será estruturado de forma a atender aos seguintes Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF):

RF01 - Monitoramento da qualidade do ar em tempo real: O sistema deve permitir que o usuário visualize os níveis de poluição do ar em tempo real, com base na localização inserida manualmente.

RF02 - Consulta de histórico de poluição: O sistema deve permitir que o usuário consulte o histórico de níveis de poluição do ar para uma determinada área, com filtros por data e localização.

RF03 - Exibição de dados de poluição de maneira gráfica: O sistema deve exibir os níveis de poluição usando gráficos simples, como gráficos de barras ou linhas, para facilitar a visualização dos dados.

RF04 - Filtragem de dados por data e localização: O sistema deve permitir que o usuário filtre os dados de poluição por período (dias, semanas, meses) e por localização específica inserida manualmente.

RF05 - Relatório comparativo de poluição: O sistema deve permitir que o usuário compare os níveis de poluição de duas ou mais localizações, exibindo a comparação em um formato gráfico ou tabelar.

RF06 - Página informativa sobre os impactos da poluição: O sistema deve conter uma página com informações sobre os impactos da poluição na saúde e meio ambiente, utilizando dados de fontes confiáveis.

RNF01 - Disponibilidade: O sistema deve estar disponível 99% do tempo, com exceção de manutenções programadas, para garantir o acesso contínuo às informações de poluição.

RNF02 - Desempenho: O sistema deve carregar as informações em até 3 segundos após a solicitação, garantindo uma boa experiência de usuário em navegadores web.

RNF03 - Segurança de Dados: O sistema deve proteger as interações com o usuário, garantindo a privacidade das informações inseridas, como a localização manual.

RNF04 - Escalabilidade: O sistema deve ser capaz de lidar com um número crescente de usuários simultâneos sem degradação no desempenho.

RNF05 - Compatibilidade com navegadores: O sistema deve ser compatível com os principais navegadores (Chrome, Firefox, Edge, Safari), garantindo o suporte para uma ampla gama de usuários.

RNF06 - Usabilidade: O sistema deve apresentar uma interface simples e intuitiva, com foco em fácil navegação, principalmente para usuários com pouca experiência em tecnologia. RF01 – Monitorar a qualidade do ar em tempo real

2.2 Atributos de Qualidade Prioritários

Os principais atributos de qualidade priorizados são:

Segurança: Implementação de camadas de segurança para proteger a privacidade dos dados dos usuários.

Usabilidade: Interface amigável e acessível, garantindo uma experiência intuitiva para usuários com pouca experiência.

Escalabilidade: Estrutura modular para suportar crescimento no número de usuários e manutenibilidade do sistema.

3. Representação da Arquitetura

3.1 Diagrama da Arquitetura Geral do Sistema

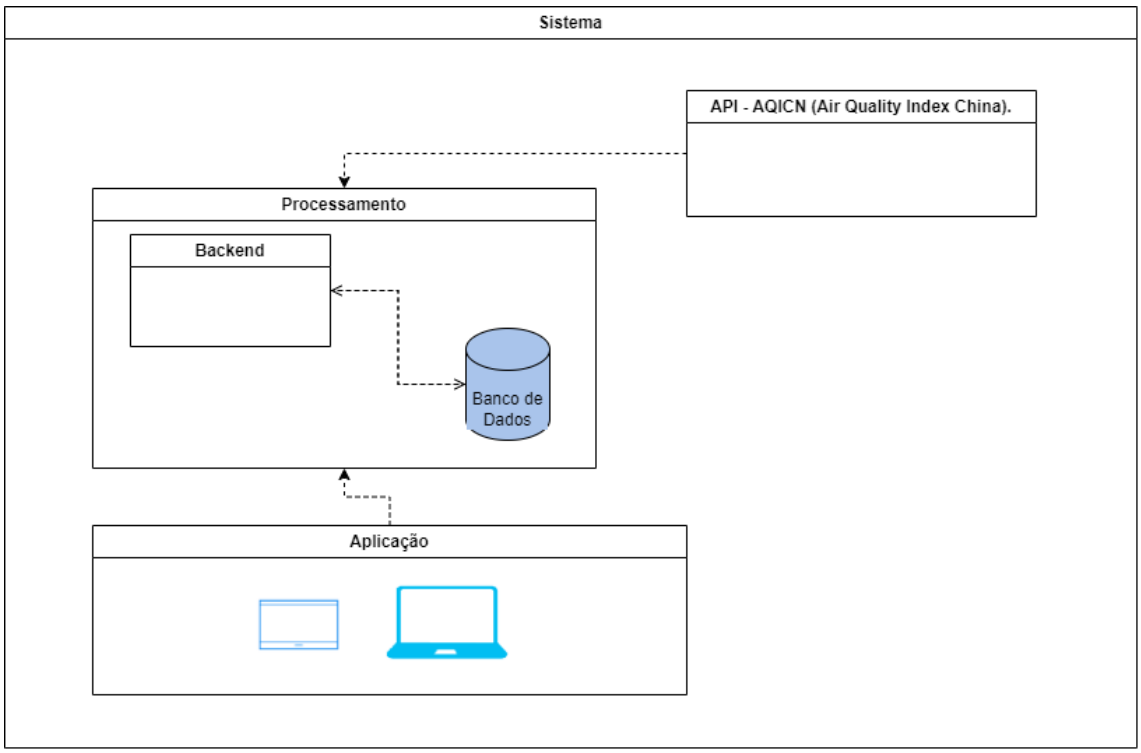


Diagrama de Arquitetura, Lucidchart

3.2 Diagrama de Comunicação

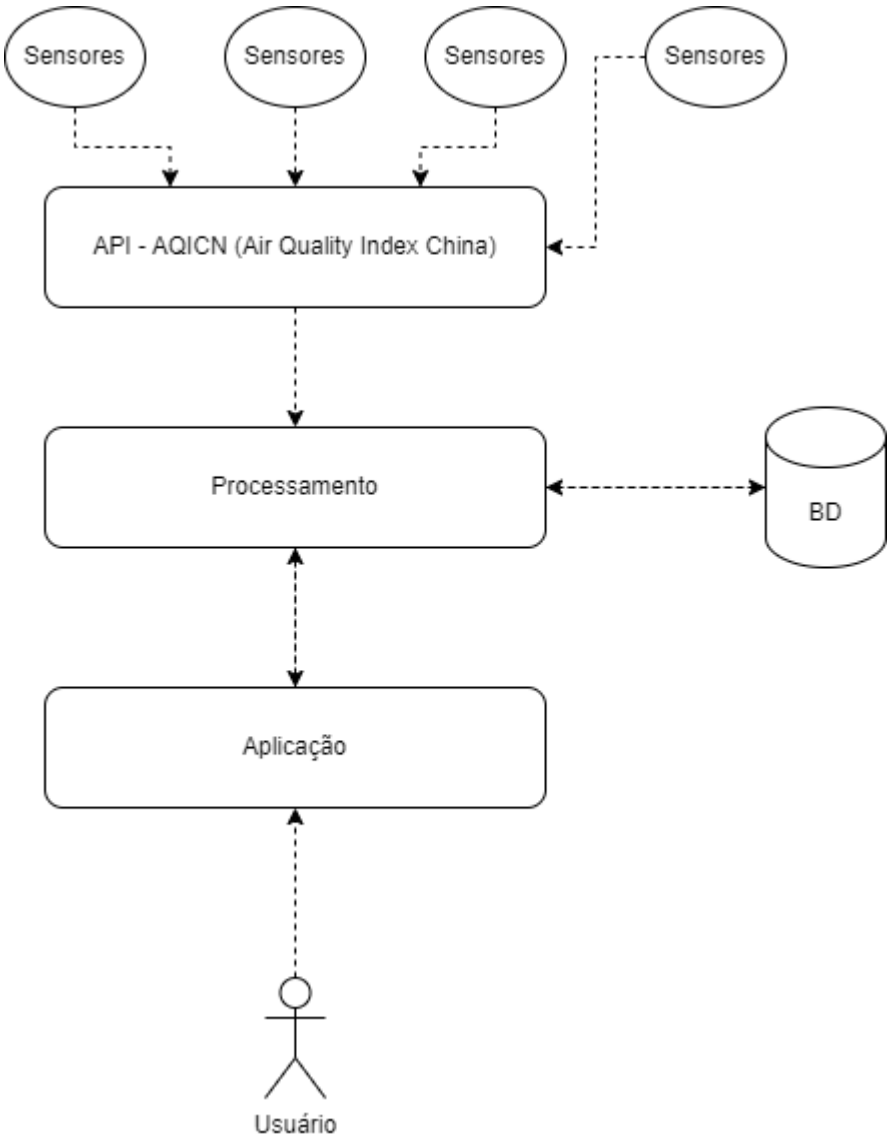


Diagrama de Comunicação - Lucidchart

3.3 Padrões Arquiteturais Adotados

O sistema utilizará uma arquitetura híbrida composta por:

Cliente-Servidor: Para a comunicação entre o frontend (cliente) e o backend (servidor).

REST: Para definição dos métodos de comunicação entre cliente e servidor, permitindo interação leve e eficiente.

MVC (Model-View-Controller): Para organizar as funcionalidades do sistema e manter a separação entre lógica de negócios, interface e dados.

3.4 Componentes Principais

Os principais componentes do sistema são:

Frontend (Cliente): Responsável por exibir a interface e interagir com o usuário. Será implementado em HTML/CSS e JavaScript, utilizando bibliotecas gráficas como Chart.js para exibir os gráficos de poluição.

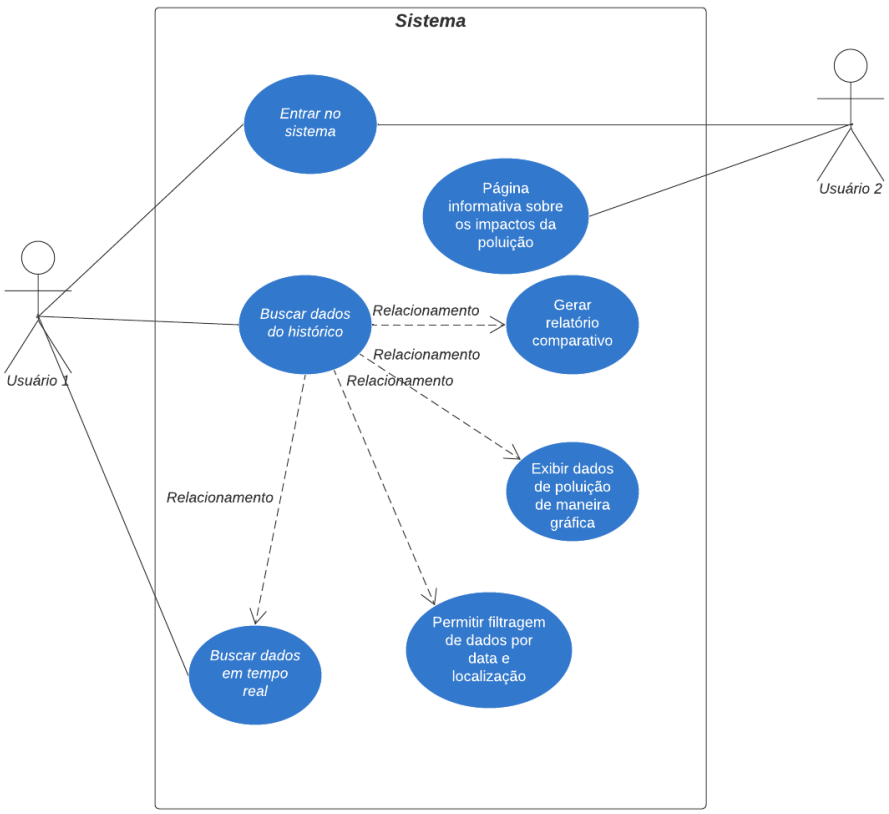
Backend (Servidor): Implementado em Node.js com Express.js, será responsável por tratar as requisições do frontend e acessar os dados de poluição.

Banco de Dados: Utilização de um banco de dados relacional (PostgreSQL) para armazenar os dados históricos e em tempo real.

API Externa: Conectada ao sistema para obter dados de poluição em tempo real, como o AQICN (Air Quality Index China).

4. Visões Arquiteturais (Em construção)

4.1 Visão de Casos de Uso



Recurso: Lucidchart

4.2 Visão Lógica

A visão lógica descreve os componentes principais do sistema e suas interações:

Frontend: Envia solicitações ao backend para obter dados e exibe gráficos e informações ao usuário.
Backend: Processa solicitações, acessa o banco de dados e retorna informações ao frontend.

Banco de Dados: Armazena dados de poluição e informações de localização.

4.3 Visão de Desenvolvimento

Esta visão detalha a estrutura do sistema em termos de organização de código e camadas:

Camada de Controle: Gerencia as interações entre frontend e backend, tratando requisições HTTP.

Camada de Negócios: Implementa as regras de negócio, como filtragem de dados e comparação de localizações.

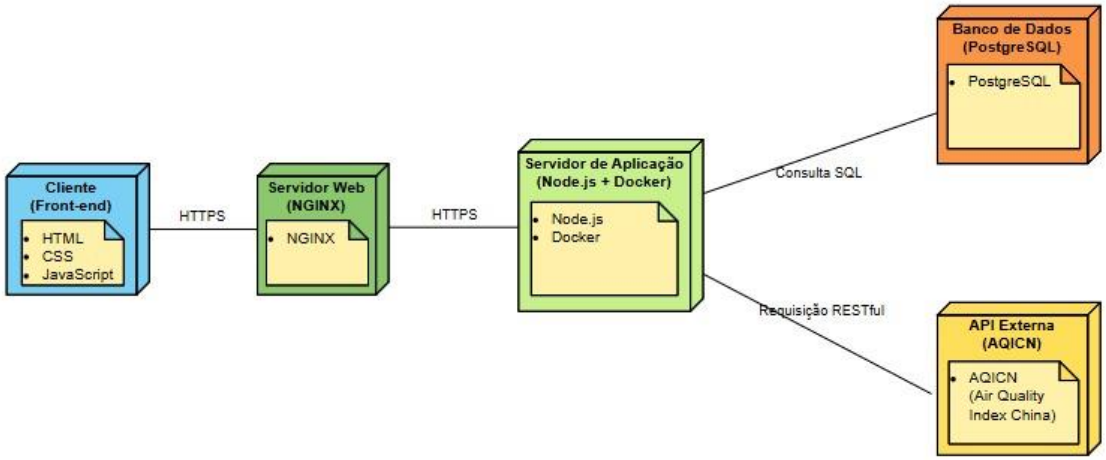
Camada de Dados: Gerencia a comunicação com o banco de dados e com a API externa.

4.4 Visão Física

A Visão Física do MediAr pode ser descrita como um sistema organizado em camadas que separa responsabilidades e garante desempenho, segurança e escalabilidade. O frontend, hospedado em um servidor web com NGINX, representa a interface do usuário, acessada via navegador e desenvolvida com tecnologias como HTML, CSS e JavaScript. Esse frontend se comunica com o backend utilizando protocolos seguros, como HTTPS, garantindo que todas as interações sejam criptografadas.

O backend é implementado em Node.js com o framework Express.js, operando dentro de um contêiner Docker. Esse contêiner facilita a implantação e permite que o sistema seja escalável conforme a demanda aumenta. O backend realiza as operações lógicas do sistema, interage com o banco de dados e acessa uma API externa, como o AQICN, para obter dados de qualidade do ar em tempo real.

O banco de dados, PostgreSQL, é usado para armazenar os dados históricos e em tempo real sobre a qualidade do ar. Ele é acessado diretamente pelo backend e está protegido por uma rede segura.



Recurso: Visual Paradigm

4.5 Visão de Segurança

A visão de segurança define as medidas para garantir a proteção dos dados dos usuários:

Criptografia SSL: Para comunicação segura entre cliente e servidor.

Autenticação: Mecanismos para verificar a integridade das solicitações.

Proteção de Dados: Uso de técnicas para anonimizar e proteger a localização inserida pelos usuários.

5. Decisões Arquiteturais

5.1 Decisões Tomadas

Arquitetura Cliente-Servidor: Escolhida para garantir separação de responsabilidades e escalabilidade.

REST: Para uma comunicação leve e eficiente entre componentes.

Uso de PostgreSQL: Escolhido pela robustez e capacidade de lidar com grandes volumes de dados históricos.

5.2 Decisões Alternativas e Justificativas

Arquitetura Monolítica: Rejeitada devido à falta de flexibilidade e dificuldade em escalar componentes individualmente.

WebSockets: Considerado para atualizações em tempo real, mas descartado para simplificar a implementação inicial.

6. Conclusão