INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA

ALBERT SILVA DE JESUS

DESAFIO INTEGRADOR

Santo Antônio de Jesus - Bahia

2020

Justificativas Técnicas

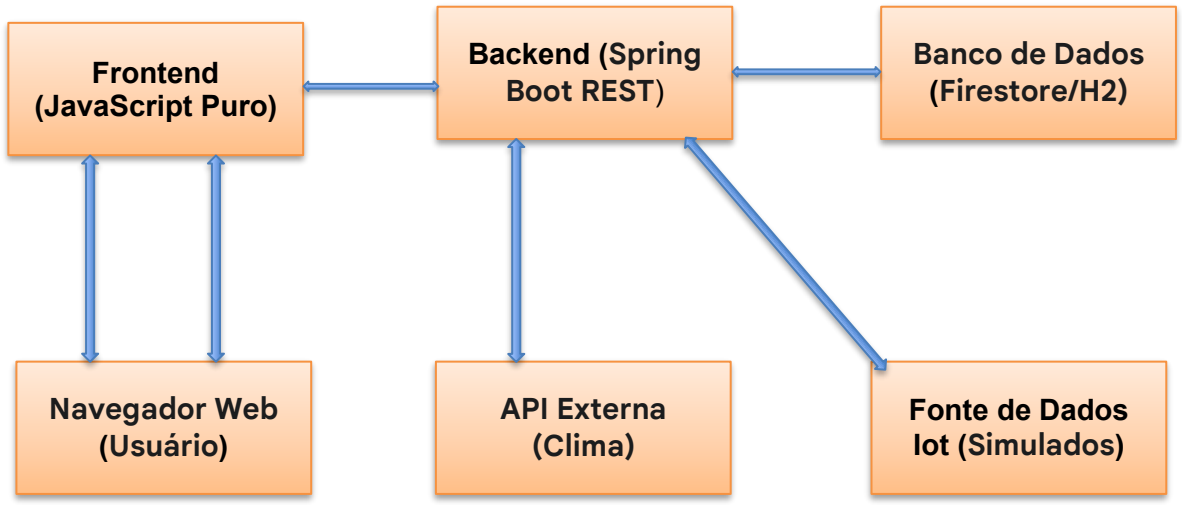
1. Introdução

Este documento apresenta as justificativas técnicas para a arquitetura e as decisões de implementação do projeto de simulação de um sistema de Internet das Coisas (IoT) focado em agricultura inteligente. O objetivo principal do projeto é demonstrar a comunicação eficiente entre um frontend e um backend para monitorar e visualizar dados de sensores em tempo real.

2. Arquitetura do Sistema

A arquitetura do projeto segue um modelo de três camadas, focado na separação de responsabilidades para garantir escalabilidade e manutenibilidade.

Diagrama de Arquitetura:



- **Frontend:** A interface de usuário (UI) é desenvolvida em JavaScript puro, HTML e CSS. Responsável por capturar a interação do usuário, fazer requisições à API do backend e exibir os dados de forma clara e intuitiva.
- **Backend:** Construído com Spring Boot, atua como um servidor de aplicações e uma API REST. É o ponto central de processamento, responsável por receber requisições do frontend, interagir com o banco de dados e, em alguns casos, com APIs externas.
- **Banco de Dados:** A persistência de dados é gerenciada por bancos de dados relacionais, utilizando PostgreSQL no ambiente de desenvolvimento e H2 para testes.
-

3. Justificativas

3.1. Justificativa para o Backend (Spring Boot)

A escolha do **Spring Boot** como tecnologia de backend foi baseada em suas fortes características e no alinhamento com os requisitos do projeto.

- **Produtividade e Convenção sobre Configuração:** O Spring Boot adota uma abordagem de convenção sobre configuração, o que reduz significativamente o tempo de desenvolvimento. As dependências e configurações são gerenciadas de forma inteligente, permitindo que o foco seja na lógica de negócio, não em burocracia.
- **Ecossistema Robusto:** O Spring Framework oferece um vasto ecossistema de módulos (Spring Data, Spring Security, etc.) que facilitam a integração com bancos de dados e a implementação de recursos de segurança e outras funcionalidades.
- **Arquitetura de Microserviços:** Spring Boot é a escolha ideal para o desenvolvimento de microserviços. Para um sistema IoT, onde cada serviço (sensores, alertas, clima) pode ser desacoplado, essa abordagem garante que o sistema seja robusto e fácil de escalar.
- **Tratamento de Dados Assíncronos:** O Spring Boot é otimizado para lidar com requisições HTTP de forma assíncrona e não bloqueante, o que é crucial para uma aplicação que lida com dados em tempo real e de múltiplos clientes.
-

3.2. Justificativa para o Frontend (JavaScript Puro)

O desenvolvimento do frontend com **JavaScript, HTML e CSS puros** foi uma decisão estratégica para este projeto de simulação.

- **Leveza e Desempenho:** Ao não utilizar um framework pesado como React ou Angular, a aplicação se torna extremamente leve, o que resulta em um carregamento rápido e uma experiência de usuário fluida. Para um painel de monitoramento que precisa ser acessado rapidamente, essa é uma grande vantagem.
- **Foco na Comunicação:** A abordagem com JS puro permite um foco maior na lógica de comunicação com a API REST. O uso da API fetch demonstra de forma clara e direta como as requisições assíncronas são realizadas, o que é um ponto de aprendizado crucial para desenvolvedores.
- **Compreensão dos Fundamentos:** Esta escolha força a compreensão dos fundamentos do desenvolvimento web, como a manipulação do DOM (document.getElementById), o tratamento de eventos e o gerenciamento de estados sem o auxílio de bibliotecas complexas.

3.3. Justificativa para o Banco de Dados (PostgreSQL e H2)

- A escolha de utilizar o PostgreSQL para o ambiente de desenvolvimento e o H2 para testes é uma prática de desenvolvimento moderna e eficiente, baseada em perfis de aplicação.
- PostgreSQL (Desenvolvimento): Sendo um dos bancos de dados relacionais mais robustos e utilizados em ambientes de produção, o PostgreSQL é a escolha ideal para o desenvolvimento. Ele oferece um conjunto completo de funcionalidades e garante que o ambiente de desenvolvimento seja o mais próximo possível do ambiente de produção, evitando surpresas indesejadas.
- H2 (Testes): O H2 é um banco de dados in-memory, leve e rápido. Ele é perfeito para testes automatizados (unitários e de integração), pois as bases de dados são criadas e destruídas rapidamente a cada ciclo de testes, garantindo que os testes sejam independentes e eficientes. A sua portabilidade dispensa a necessidade de um servidor de banco de dados externo durante a execução dos testes.
- Perfis de Aplicação (application.properties): A separação das configurações em arquivos como application-dev.properties e application-test.properties é a chave para essa estratégia. O Spring Boot, através de perfis de ambiente, carrega a configuração correta para cada contexto (desenvolvimento ou teste). Isso permite que a mesma base de código funcione em diferentes ambientes sem a necessidade de alterações manuais.
-

4. Justificativas da Escolha dos Protocolos de Comunicação

Justificativa da Escolha dos Protocolos de Comunicação

Para garantir uma comunicação eficiente e robusta em diferentes cenários, foram selecionados três protocolos de comunicação, cada um com um papel específico no sistema. A escolha foi baseada em critérios como latência, consumo de recursos e confiabilidade.

HTTP REST

Justificativa: O HTTP REST foi utilizado para a comunicação cliente-servidor, especialmente para endpoints que requerem interação direta do usuário, como consultar dados históricos (GET /api/sensores) e alterar o estado de alertas (PUT /api/sensores/alertas/status/{status}).

Características:

Latência: Moderada. Embora a latência possa ser maior devido ao cabeçalho verboso do HTTP, é perfeitamente aceitável para interações não-críticas e pontuais.

Consumo: Alto. O HTTP é um protocolo mais "pesado" em termos de cabeçalho, o que o torna menos ideal para dispositivos IoT com recursos limitados, mas é a escolha padrão para APIs web.

Confiabilidade: Alta. O HTTP possui mecanismos de confiabilidade (como códigos de status) que garantem que as requisições e respostas sejam entregues e processadas corretamente.

MQTT (Message Queuing Telemetry Transport)

Justificativa: O MQTT é o protocolo ideal para a comunicação entre a aplicação e dispositivos IoT (como a simulação de sensores de temperatura), especialmente em redes com baixa largura de banda ou alta latência.

Características:

Latência: Baixa. Seu cabeçalho mínimo (a partir de 2 bytes) e a natureza orientada a eventos (publish/subscribe) resultam em uma latência muito baixa, ideal para mensagens em tempo real.

Consumo: Baixo. É um protocolo leve, projetado para ser eficiente em consumo de energia e de dados, o que o torna a escolha preferida para dispositivos de campo.

Confiabilidade: Moderada a Alta. O MQTT oferece três níveis de Quality of Service (QoS), que permitem controlar a confiabilidade da entrega das mensagens. No nosso caso, o QoS mais adequado pode ser configurado para o cenário de monitoramento.

AMQP (Advanced Message Queuing Protocol)

Justificativa: O AMQP foi escolhido para a comunicação interna do sistema, agindo como uma ponte de dados confiável entre diferentes módulos e serviços da aplicação. Ele é ideal para sistemas que exigem processamento assíncrono e garantia de entrega de mensagens.

Características:

Latência: Moderada. A latência é um pouco maior que a do MQTT devido à sua natureza mais robusta e complexa, mas é otimizada para alto rendimento e processamento em lote.

Consumo: Moderado. Possui um overhead maior que o MQTT, mas oferece um modelo mais flexível e seguro, adequado para a comunicação entre serviços de back-end.

Confiabilidade: Altíssima. A arquitetura de troca de mensagens do AMQP, com filas, exchanges e roteamento complexo, garante a entrega da mensagem mesmo em caso de falhas, o que é crucial para dados de alerta.

Arquitetura Híbrida e Padrões de Comunicação

A aplicação foi projetada com uma arquitetura híbrida que combina processamento local (Edge), intermediário (Fog) e remoto (Cloud) para otimizar o fluxo de dados, reduzir a latência e aumentar a confiabilidade.

Camadas da Arquitetura

Edge (Processamento Local):

Função: Coleta inicial de dados e processamento leve e imediato. O objetivo é tomar decisões rápidas sem depender da nuvem.

Componentes: O SensorScheduler e a lógica de verificação de alertas no SensorService representam o processamento de borda. Eles geram e analisam os dados no local de origem, decidindo se um alerta deve ser gerado antes de qualquer comunicação externa.

Fog (Processamento Intermediário):

Função: Agir como uma ponte e um nó de agregação entre a borda e a nuvem. Ele pode filtrar dados, converter protocolos ou armazenar dados temporariamente.

Componentes: A classe MqttToAmqpBridge atua como a camada Fog. Ela recebe dados de uma fonte de borda (via MQTT) e os encaminha de forma confiável para o processamento na nuvem (via AMQP).

Cloud (Processamento Remoto):

Função: Oferecer recursos de processamento massivo, armazenamento de dados em grande escala e integração com serviços externos.

Componentes: O SensorService, o SensorDataRepository e o novo WeatherService formam a camada Cloud. Eles persistem todos os dados, realizam análises complexas, gerenciam o histórico e se integram com APIs externas como a do OpenWeather.

Padrões de Comunicação

Publish/Subscribe: Adotado para a comunicação dos sensores. Os dados são publicados em tópicos (via MQTT e AMQP) para que qualquer sistema interessado possa consumi-los.
Request/Response: Utilizado para a comunicação com APIs externas, como a API do OpenWeather. A nossa aplicação envia uma requisição e espera por uma resposta.

5. Conclusão

A arquitetura e as escolhas técnicas deste projeto de simulação de IoT para agricultura inteligente foram cuidadosamente planejadas para criar um sistema funcional, robusto e escalável. A combinação do backend em Spring Boot com o frontend em JavaScript puro fornece uma base sólida para a interação com o usuário, enquanto a estratégia de banco de dados com PostgreSQL e H2 assegura a persistência de dados em diferentes ambientes.

A decisão mais estratégica, no entanto, foi a adoção de uma arquitetura híbrida que integra múltiplos protocolos de comunicação (HTTP, MQTT, AMQP). Essa abordagem garante que cada componente do sistema utilize o protocolo mais adequado para sua função, otimizando a latência e o consumo de recursos. Como resultado, o sistema demonstra de forma prática como a integração de diferentes tecnologias e a separação de responsabilidades podem resultar em uma solução de monitoramento eficiente e preparada para as demandas do mundo real.

Referência

DEITEL, Paul; DEITEL, Harvey. Java: como programar. Tradução de Edson Furmankiewicz. Revisão técnica de Fabio Lucchini. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

NETO, O. M. Entendendo e Dominando o Java. [S.l.]: Digerati Books, 2009.

BLOCH, Joshua. Java efetivo: as melhores práticas para a plataforma Java. Tradução de Cibelle Ravaglia. Revisão técnica de Alline Santos Ferreira. 1. ed. rev. conforme o Acordo Ortográfico da Língua Portuguesa de 2009. Rio de Janeiro: Alta Books, 2019.

ORACLE. JAVA. Disponível em: Software Java | Oracle Brasil . Acesso em 09 Jan. 2024.

ARAÚJO, Iasmin. Garbage Collector: como funciona o coletor de lixo do Java? Alura, 12 jun. 2024. Disponível em: https://www.alura.com.br/artigos/garbage-collector?srsltid=AfmBOopf6ZpSZkA8LPgSn5wETIPD5gVTdYRfIFXex6VWCwu1XumVO3bD. Acesso em: 20 jul. 2025.

CAELUM. Apostila Java para Desenvolvimento Web. Disponivel em: https://www.alura.com.br/apostila-java-web/o-que-e-java-ee#3-1-como-o-java-ee-pode-te-ajudar-a-enfrentar-problemas. Acesso em: 07 mai. 2025.

BOAGLIO, Fernando. Spring Boot: acelere o desenvolvimento de microsserviços. São Paulo: Casa do Código, 2017. Edição de Adriano Almeida e Vivian Matsui.

JUNIOR, Normandes; AFONSO, Alexandre. Produtividade no desenvolvimento de aplicações web com Spring Boot. 2. ed. São Paulo: AlgaWorks, 2017.

SPRING.IO. Why Spring. Spring by VMware Tanzu, 2025. Disponível em: https://spring.io/why-spring. Acesso em: 22 mar. 2025.