Relatório Fila Inteligente SUS

Fila Inteligente SUS

1. Resumo Executivo

O projeto **Fila Inteligente SUS** propõe uma solução tecnológica para otimizar o fluxo de atendimento em unidades do Sistema Único de Saúde (SUS). A aplicação calcula o tempo estimado de espera do paciente com base em sua posição na fila e no tempo médio de atendimento, comparando esse valor com o tempo estimado de deslocamento até a unidade de saúde.

Quando o sistema identifica que o momento ideal para deslocamento foi atingido, uma notificação é enviada automaticamente por meio de mensageria assíncrona. A solução busca reduzir aglomerações, melhorar a experiência do paciente e otimizar a organização das unidades de atendimento.

---

2. Problema Identificado

Um dos principais desafios enfrentados pelo SUS é a gestão de filas presenciais, especialmente em unidades de atenção básica e ambulatórios.

Problemas observados:

- Longo tempo de espera nas unidades
- Aglomeração de pacientes em salas de espera
- Falta de previsibilidade sobre o momento exato do atendimento
- Deslocamentos desnecessários e tempo improdutivo do paciente

A ausência de um sistema inteligente que informe o momento adequado para deslocamento contribui para sobrecarga estrutural e desconforto dos usuários.

---

3. Descrição da Solução

A solução desenvolvida consiste em um sistema backend baseado em microservices que:

1. Permite cadastrar pacientes e inseri-los em uma fila de atendimento.
2. Calcula o tempo estimado de espera com base na posição do paciente e no tempo médio de atendimento.
3. Compara o tempo estimado de espera com o tempo de deslocamento do paciente.
4. Caso o tempo de espera seja menor ou igual ao tempo de deslocamento, dispara automaticamente uma notificação.

Fluxo Simplificado

Paciente → Fila Service → RabbitMQ → Notification Service

A comunicação entre serviços ocorre de forma assíncrona por meio de mensageria, garantindo desacoplamento e escalabilidade.

---

4. Processo de Desenvolvimento

O desenvolvimento seguiu as seguintes etapas:

4.1 Brainstorming

Discussão sobre problemas reais enfrentados pelo SUS e possíveis soluções digitais.

4.2 Definição do MVP

Delimitação de um escopo mínimo viável com foco exclusivo em backend e arquitetura distribuída.

4.3 Modelagem com Event Storming

Foi utilizada a técnica de **Event Storming**, proveniente do Domain-Driven Design (DDD), para identificar os principais eventos de negócio do sistema.

Eventos identificados: - Paciente cadastrado - Paciente entrou na fila - Tempo de espera calculado - Momento de deslocamento atingido - Notificação enviada

Essa abordagem permitiu estruturar corretamente as regras de negócio antes da implementação técnica.

4.4 Arquitetura

Decisão por arquitetura baseada em microservices com comunicação via mensageria (RabbitMQ) e aplicação de conceitos de CQRS.

4.5 Implementação

- Desenvolvimento do serviço principal de fila
- Implementação de serviço independente de notificações
- Integração via RabbitMQ
- Containerização com Docker e Docker Compose

4.6 Testes

Testes realizados via Swagger (OpenAPI) e validação de envio e consumo de eventos.

---

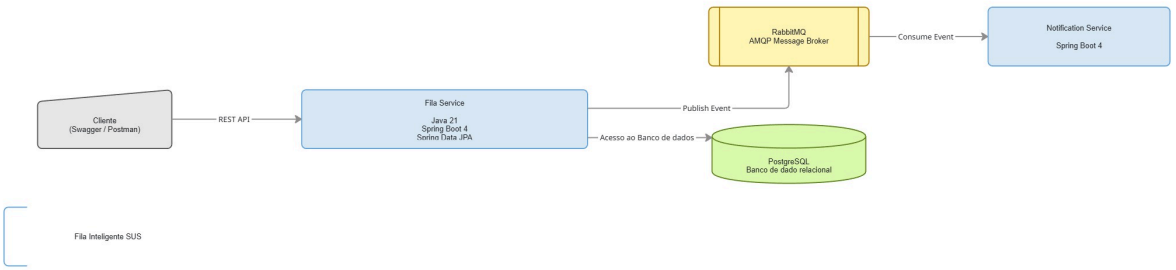
5. Detalhes Técnicos

Tecnologias Utilizadas

- Linguagem: Java 21
- Framework: Spring Boot 4
- Persistência: Spring Data JPA
- Banco de Dados: PostgreSQL
- Mensageria: RabbitMQ (AMQP)
- Documentação de API: Springdoc OpenAPI (Swagger)
- Containerização: Docker
- Orquestração: Docker Compose

Arquitetura do Sistema

Arquitetura baseada em microservices com comunicação orientada a eventos:



- **fila-service (porta 8080)**

  Responsável por cadastro, cálculo de tempo de espera e publicação de eventos.

- **notification-service (porta 8081)**

  Responsável por consumir eventos e registrar notificações.

- **RabbitMQ (porta 5672)**

  Broker de mensageria responsável pela comunicação assíncrona.

- **PostgreSQL**

  Persistência de dados do serviço de fila.

5.1 Aplicação de Event Storming e CQRS

Event Storming

Durante a fase de modelagem do sistema, foi aplicada a abordagem de **Event Storming** para mapear os eventos de negócio do domínio.

O evento central modelado foi:

> Momento de deslocamento atingido

Esse evento é responsável por disparar a publicação de uma mensagem no RabbitMQ, que será consumida pelo serviço de notificações.

A modelagem orientada a eventos permitiu alinhar o sistema ao fluxo real de atendimento do SUS, garantindo coerência entre regra de negócio e arquitetura.

CQRS (Command Query Responsibility Segregation)

O sistema aplica conceitos de **CQRS**, separando operações de escrita (Command) e leitura (Query).

Commands (Escrita): - Cadastro de paciente - Inserção na fila - Publicação de evento de notificação

Queries (Leitura): - Consulta de status da fila - Consulta de notificações enviadas - Listagem da fila atual

A responsabilidade de escrita está concentrada no `fila-service`, enquanto o `notification-service` consome eventos e disponibiliza dados para consulta.

Essa separação proporciona: - Desacoplamento entre responsabilidades - Escalabilidade independente - Melhor organização arquitetural - Maior clareza na modelagem do domínio

---

6. Links Úteis

- Repositório do projeto: https://github.com/joseleite550/fila_inteligente
- Documentação da API (Swagger): http://localhost:8080/swagger-ui/index.html

---

7. Aprendizados e Próximos Passos

Aprendizados

- Implementação prática de microservices com Java e Spring Boot
- Integração assíncrona via RabbitMQ
- Aplicação de Event Storming para modelagem de domínio
- Aplicação prática de CQRS
- Containerização e orquestração com Docker Compose

Próximos Passos

- Implementar autenticação (JWT ou OAuth2)
- Criar frontend web ou aplicativo mobile
- Implementar envio real de notificações (SMS, Push ou WhatsApp)
- Adicionar monitoramento (Prometheus/Grafana)
- Criar módulo compartilhado para DTOs comuns

- Implementar testes automatizados (JUnit + Testcontainers)

Autor(es):

José Leite, Vitor Santos