Relatório de Sistemas Distribuídosde Gerenciamento de Remédios via gRPC

Thiago Medeiros Carvalho e Carlos Eduardo Alves Ferreira- 2025/1

Junho 2025

Contents

1 Introdução

Este relatório descreve o desenvolvimento e implementação de um sistema distribuído para gerenciamento de medicamentos utilizando o protocolo gRPC, baseado no modelo RPC (Remote Procedure Call). O sistema proporciona operações remotas que facilitam o controle e a gestão do estoque de medicamentos.

2 Contextualização do Problema

Muitas farmácias e clínicas enfrentam problemas com métodos manuais ou parcialmente informatizados para controle de medicamentos. Tais abordagens podem gerar inconsistências, perda de informação e dificuldade no rastreamento de alterações. Visando resolver essas limitações, propomos uma solução distribuída baseada em RPC, mais especificamente utilizando gRPC, que permite comunicações eficientes, estruturadas e transparentes entre clientes e servidores.

3 Objetivos do Sistema

O sistema proposto busca resolver de maneira eficiente e escalável as demandas de gerenciamento e controle de medicamentos em ambientes distribuídos, destacando-se pelo uso da tecnologia gRPC para comunicação remota eficiente, robusta e transparente.

3.1 Objetivo Geral

Desenvolver e implementar um sistema distribuído robusto e escalável para gerenciamento remoto de medicamentos, fundamentado no modelo de chamadas de procedimento remoto (RPC), utilizando o protocolo gRPC para assegurar eficiência e clareza na comunicação cliente-servidor.

3.2 Objetivos Específicos

- **Desenvolver uma API RPC clara e estruturada:** Implementar interfaces bem definidas para as operações essenciais (cadastrar, listar, detalhar, atualizar, dar baixa no estoque e consultar histórico), permitindo fácil integração e utilização por aplicações clientes.
- **Utilizar armazenamento robusto e persistente:** Integrar o sistema com um banco de dados relacional (MySQL) para garantir consistência, segurança e persistência confiável das informações críticas dos medicamentos.
- **Prover uma interface web amigável e intuitiva:** Disponibilizar ao usuário final uma interface simples, direta e responsiva, facilitando o

gerenciamento de medicamentos e a interação com as funcionalidades distribuídas.

- **Assegurar comunicação eficiente com gRPC:** Explorar o uso do protocolo gRPC com serialização via Protocol Buffers para reduzir latências, aumentar desempenho e garantir a integridade das mensagens trocadas remotamente entre cliente e servidor.
- **Garantir rastreabilidade detalhada e confiável:** Implementar mecanismos que permitam manter histórico detalhado das alterações realizadas sobre os medicamentos, fornecendo visibilidade clara das operações realizadas e das condições de estoque ao longo do tempo.
- **Facilitar escalabilidade e manutenção do sistema:** Adotar uma arquitetura modular e distribuída, permitindo crescimento horizontal (mais clientes e servidores) e simplificando futuras atualizações e manutenções no sistema.
- **Explorar o papel dos stubs RPC:** Demonstrar claramente o papel dos stubs gRPC na abstração da comunicação remota, facilitando a integração entre sistemas heterogêneos e reduzindo a complexidade do desenvolvimento.
- **Aplicar conceitos fundamentais de sistemas distribuídos:** Utilizar o projeto como oportunidade prática para aprofundar conceitos acadêmicos fundamentais sobre comunicação inter-processos, modelos cliente/servidor, tolerância a falhas e consistência em sistemas distribuídos.

4 Arquitetura do Sistema

Para atender às necessidades específicas do gerenciamento remoto e eficiente de medicamentos, foi adotada uma arquitetura distribuída baseada no modelo cliente-servidor, complementada pelo uso intensivo da tecnologia de comunicação via RPC (Remote Procedure Call), mais especificamente através do protocolo gRPC.

4.1 Modelo Cliente/Servidor

O sistema foi projetado com uma arquitetura cliente-servidor claramente definida, onde cada componente desempenha papéis distintos, porém complementares:

- **Servidor (Java/Spring Boot)**: Responsável pela lógica de negócio, segurança das informações, persistência dos dados em um banco relacional (MySQL) e pela exposição dos serviços via API gRPC. Este componente concentra o processamento das requisições e garante a consistência e integridade das operações executadas. Aqui, conceitos de **processos**

e threads são explorados, permitindo ao servidor tratar múltiplas solicitações simultaneamente de forma eficiente, utilizando threads gerenciadas pelo framework Spring Boot (tópico visto na aula de 24/03/2025).

- **Cliente (Node.js/Express)**: O cliente consome serviços remotos através das chamadas RPC para realizar operações como cadastro, consulta e atualização de medicamentos, oferecendo uma interface REST para comunicação com uma interface web amigável. A separação clara entre cliente e servidor facilita a manutenção e a escalabilidade do sistema, características fundamentais de arquiteturas distribuídas (tema discutido na aula de Arquiteturas, em 19/03/2025).

A separação explícita entre cliente e servidor torna o sistema mais modular e facilita a aplicação de boas práticas relativas à **replicação e consistência** dos dados, essenciais para garantir alta disponibilidade e tolerância a falhas (tópicos abordados em aulas de 07/05/2025 a 26/05/2025).

4.2 Comunicação via RPC (gRPC)

O uso do protocolo gRPC como tecnologia de comunicação é um elemento-chave para o sucesso do projeto. O gRPC oferece uma forma eficiente, robusta e transparente de comunicação entre processos distribuídos (tópico central discutido entre as aulas de 26/03/2025 e 31/03/2025).

O protocolo gRPC utiliza a serialização de mensagens através de Protocol Buffers (protobuf), o que garante comunicação eficiente devido à sua leveza e desempenho superior em comparação a formatos tradicionais como JSON. Além disso, a abordagem RPC permite que métodos remotos sejam chamados como se fossem locais, abstraindo os detalhes mais complexos da comunicação interprocessos e facilitando a programação distribuída.

Entre as vantagens da escolha por RPC/gRPC, destacam-se:

- **Transparência de comunicação**: O desenvolvedor pode realizar chamadas remotas sem precisar lidar diretamente com a complexidade das comunicações de rede, pois o gRPC trata esses detalhes automaticamente.
- **Eficiência e desempenho**: Devido à utilização de Protocol Buffers, o gRPC reduz significativamente a latência e a sobrecarga na rede, algo especialmente crítico para aplicações que demandam grande volume de requisições simultâneas.
- **Facilidade na escalabilidade**: O modelo RPC, especialmente quando implementado via gRPC, facilita a adição de novos métodos e funcionalidades, permitindo crescimento horizontal com menor esforço de desenvolvimento e manutenção.
- **Suporte integrado à segurança e autenticação**: O gRPC oferece suporte integrado para segurança através de TLS e autenticação baseada em tokens JWT, aspectos fundamentais da segurança em sistemas distribuídos (tópico discutido nas aulas de 28/05/2025 e 02/06/2025).

4.3 Stubs gRPC

Um dos aspectos mais importantes da utilização do gRPC é a presença dos stubs, componentes críticos que implementam um papel de proxy e abstraem a complexidade da comunicação entre cliente e servidor.

No contexto deste projeto, os stubs gerados automaticamente a partir dos arquivos protobuf atuam conforme descrito abaixo:

- **Stub no lado Cliente**: Responsável por receber chamadas locais dos métodos desejados, serializar os parâmetros utilizando Protocol Buffers e enviar requisições para o servidor através da rede. Isso proporciona uma experiência semelhante à programação local, reduzindo a complexidade inerente à comunicação remota.
- **Stub no lado Servidor**: Recebe as requisições remotas, desserializa as mensagens recebidas, invoca o método adequado com os parâmetros corretos e, após a execução, serializa a resposta para envio de volta ao cliente.

Esses componentes garantem a consistência das chamadas remotas e simplificam o desenvolvimento ao ocultar detalhes de baixo nível da comunicação, como gerenciamento de sockets, serialização/desserialização e controle de erros.

Além disso, os stubs auxiliam na padronização das interfaces e métodos expostos, promovendo uma clara separação entre interfaces e implementações, característica fundamental abordada nas aulas de Comunicação (26/03/2025 - 31/03/2025) e Nomeação (02/04/2025 - 07/04/2025), pois facilitam a identificação e chamada de procedimentos remotos através de nomes padronizados e claros.

4.4 Vantagens específicas para este projeto

O projeto aproveita integralmente a potencialidade do RPC/gRPC, destacando especialmente os seguintes pontos:

- A comunicação eficiente e padronizada permitiu integração transparente entre o cliente (Node.js) e o servidor (Java/Spring), tecnologias distintas que poderiam gerar complexidade adicional sem uma abstração adequada.
- O uso de stubs facilitou imensamente o processo de desenvolvimento e manutenção, reduzindo tempo gasto na depuração e em questões relacionadas à comunicação direta entre cliente e servidor.
- O desempenho oferecido pelo protocolo gRPC se mostrou essencial na manipulação rápida e confiável de operações frequentes e críticas, como cadastro, consulta e controle de estoques.
- A arquitetura modularizada e baseada em RPC proporcionou escalabilidade facilitada, permitindo adicionar facilmente novos servidores ou clientes, conforme necessário.

5 Descrição Detalhada da API gRPC

Segue a descrição dos principais métodos da API gRPC:

- **Cadastrar**: recebe informações sobre o remédio e retorna o objeto criado.
- **Listar**: retorna uma lista resumida dos remédios cadastrados.
- **Detalhar**: retorna detalhes completos de um remédio específico.
- **Atualizar**: recebe alterações de informações do remédio e retorna o objeto atualizado.
- **DarBaixaEstoque**: decrementa a quantidade do estoque e retorna o estado atualizado.
- **ConsultarHistorico**: retorna um histórico detalhado das alterações efetuadas no remédio.

6 Modelo de Dados

O modelo de dados do sistema foi cuidadosamente projetado para garantir uma persistência eficiente, estruturada e que facilite operações rápidas e robustas, respeitando princípios discutidos ao longo da disciplina, como replicação, consistência e sincronização.

As entidades principais, armazenadas no banco de dados relacional MySQL, são:

- **remedio**:
  - **id**: identificador único do medicamento (chave primária, autoincrementada).
  - **nome**: nome comercial do medicamento.
  - **via**: via de administração (ENUM com valores ORAL, VENOSO, RETAL, NASAL, INTRAMUSCULAR).
  - **lote**: código identificador do lote.
  - **quantidade**: quantidade disponível em estoque.
  - **validade**: data de vencimento do medicamento.
  - **laboratorio**: laboratório fabricante (ENUM com valores ACHE, MEDLEY).
  - **ativo**: estado ativo ou inativo do medicamento.
- **historico_alteracao**:
  - **id**: identificador único da alteração (chave primária, autoincrementada).

- **remedio_id**: identificador do medicamento alterado (chave estrangeira).
- **estado_antigo**: representação do medicamento antes da alteração (armazenado em formato JSON).
- **estado_novo**: representação do medicamento após alteração (JSON).
- **tipo_alteracao**: descrição sucinta da operação realizada (e.g., cadastro, atualização, baixa de estoque).
- **data_alteracao**: data e hora da alteração.

Este modelo suporta plenamente os requisitos do sistema, permitindo consultas rápidas, auditoria detalhada de alterações e operações eficientes, em linha com os princípios de **consistência** e **replicação**, que foram amplamente debatidos na disciplina entre 07/05/2025 e 14/05/2025.

7 Fluxo de Requisições via RPC

O fluxo das requisições no sistema distribuído envolve múltiplas camadas, seguindo claramente conceitos da disciplina como comunicação remota, nomeação e sincronização. O fluxo básico ocorre da seguinte forma:

1. O cliente envia requisições HTTP REST ao servidor Node.js (Express), utilizando métodos HTTP padrão (POST, GET, PUT, DELETE).
2. O servidor Express interpreta essas requisições REST e realiza chamadas RPC ao servidor Java através dos stubs gRPC gerados automaticamente. Esses stubs encapsulam detalhes de comunicação, serialização e envio das mensagens.
3. O servidor Java recebe as chamadas RPC, desserializa os dados utilizando os stubs correspondentes, executa a lógica de negócio solicitada, acessa o banco MySQL para consultas ou atualizações e responde serializando novamente as informações através do protocolo gRPC.
4. O stub gRPC no cliente recebe a resposta serializada do servidor, desserializa-a, e então converte-a em uma resposta HTTP JSON ao front-end, finalizando o ciclo da requisição com sucesso.

Este fluxo aproveita plenamente a transparência e eficiência proporcionada pelo uso dos stubs RPC, permitindo comunicação robusta e segura, conforme explorado detalhadamente nas aulas sobre comunicação (26/03/2025 – 31/03/2025) e nomeação (02/04/2025 – 07/04/2025).

8 Casos de Uso e Testes Realizados

O sistema foi submetido a diversos testes abrangentes para garantir que ele atende plenamente aos requisitos propostos, em termos de desempenho, robustez e tratamento de falhas. Os principais testes realizados incluem:

- **Cadastro e validação de campos obrigatórios:** Verificação de campos obrigatórios e consistência dos tipos de dados para garantir integridade na criação dos medicamentos.

  swift Copiar Editar

- **Listagem e detalhes de medicamentos:** Garantir recuperação correta e consistente das informações armazenadas, verificando especialmente operações simultâneas para testar a correta aplicação do conceito de processos e threads (24/03/2025).

- **Atualização de informações:** Verificação da persistência correta de atualizações, mantendo consistência entre o estado armazenado e apresentado ao usuário final.

- **Controle e baixa de estoque com tratamento de exceções (estoque insuficiente):** Avaliação da robustez do sistema ao lidar com situações adversas, como tentativas de redução do estoque abaixo do permitido, explorando conceitos de sincronização (09/04/2025 – 28/04/2025) e tolerância a falhas (19/05/2025 – 26/05/2025).

- **Histórico detalhado de alterações:** Avaliação completa da funcionalidade de auditoria e rastreabilidade, assegurando que todas as alterações estejam corretamente registradas e acessíveis.

Esses testes demonstraram o alto grau de maturidade do sistema desenvolvido, refletindo diretamente os aprendizados práticos dos tópicos discutidos ao longo da disciplina.

9 Conclusão

O desenvolvimento do sistema distribuído utilizando o modelo RPC através do protocolo gRPC permitiu abordar na prática muitos dos principais tópicos explorados ao longo da disciplina Sistemas Distribuídos da UFRRJ, tais como arquiteturas distribuídas, processos e threads, comunicação inter-processos, sincronização, tolerância a falhas, consistência e segurança.

A implementação realizada comprovou as vantagens inerentes ao uso do protocolo gRPC, como eficiência, clareza na comunicação remota, abstração proporcionada pelos stubs RPC e facilidade na escalabilidade. O sistema final apresenta uma arquitetura robusta, modular e escalável, totalmente capaz de suportar ambientes reais e demandas práticas na gestão de medicamentos.

Como sugestões para evoluções futuras, propõe-se a implementação de mecanismos mais robustos de segurança e autenticação (baseados em tokens JWT ou OAuth2, temas abordados em aula no período de 28/05/2025 – 02/06/2025), a integração com serviços de nuvem para suportar um número maior de usuários simultâneos e a ampliação dos mecanismos de persistência e auditoria histórica.

Desta forma, o projeto não apenas cumpre os objetivos acadêmicos propostos, mas também fornece uma base prática sólida para futuras pesquisas e implementações mais complexas no campo de sistemas distribuídos.