# **ReRelatório Técnico: Arquitetura e Modelagem de Domínio para o Sistema FreePharma**

# **Este relatório detalha a arquitetura e a modelagem de domínio para o sistema FreePharma, com foco na implementação de uma arquitetura limpa (Clean Architecture) e no padrão CQRS (Command Query Responsibility Segregation) utilizando Spring Boot 3. O objetivo é fornecer um plano técnico robusto que garanta a escalabilidade, solidez e manutenibilidade do sistema, alinhado aos requisitos fiscais e operacionais de farmácias.**

## **1. Visão Geral do Sistema FreePharma e Contexto Arquitetural**

# **Esta seção estabelece o entendimento fundamental do sistema FreePharma, seu propósito central, os requisitos que o impulsionam e as escolhas arquiteturais que guiarão seu desenvolvimento.**

### **1.1. Propósito e Escopo do FreePharma**

# **O sistema FreePharma é concebido como uma solução abrangente de gestão fiscal, especificamente adaptada para farmácias. Seu objetivo primordial é automatizar a organização, leitura e análise de notas fiscais eletrônicas (NF-e e NFC-e), visando aprimorar a segurança fiscal e otimizar os processos de escrituração e declaração de obrigações tributárias, como Imposto de Renda (IR), SPED Fiscal e Sintegra.**

# **As funcionalidades essenciais do FreePharma incluem a importação e gestão centralizada de notas fiscais, a detecção automática de inconsistências fiscais – como divergências de valores, erros de classificação tributária (NCM, CFOP, CST) e duplicidades – e a geração de relatórios inteligentes. Estes relatórios são projetados para evidenciar irregularidades, identificar oportunidades de correção e auxiliar na tomada de decisões contábeis e fiscais. A natureza do sistema, focada em "segurança fiscal" e "eficiência no processo de escrituração e declaração de obrigações tributárias", ressalta que a integridade, precisão e conformidade dos dados não são meras funcionalidades, mas sim requisitos fundamentais e inegociáveis. Isso implica que o design do sistema deve priorizar mecanismos robustos de validação, auditabilidade e tratamento de erros para prevenir repercussões legais e financeiras para as farmácias. A escolha do PostgreSQL como sistema de gerenciamento de banco de dados relacional apoia inerentemente a integridade transacional necessária para um sistema fiscal.**

### **1.2. Requisitos Funcionais e Não Funcionais Chave**

# **Os requisitos do sistema FreePharma, tanto funcionais quanto não funcionais, exercem uma influência direta e significativa sobre as decisões de design arquitetural.**

# **Os principais Requisitos Funcionais (RFs) incluem:**

# **RF001 - Cadastro de Farmácia: Permite registrar informações essenciais da farmácia, incluindo dados fiscais e certificado digital.**

# **RF002 - Cadastro de Unidades (Filiais, Depósitos): Gerencia unidades vinculadas à matriz, com controle de estoque e obrigações específicas.**

# **RF003 - Cadastro de Usuário Administrador: Permite criar e gerenciar usuários com permissões de acesso.**

# **RF004 - Gerenciamento de Fornecedores: Habilita o cadastro e a gestão de fornecedores, facilitando a associação de notas fiscais.**

# **RF005 - Gerenciamento de Produtos: Permite o cadastro e a gestão de produtos, incluindo dados fiscais e de validade.**

# **RF006 - Gerenciamento de Estoque: Atualiza estoques em tempo real a partir de NF-e e emite alertas.**

# **RF007 - Importação de Notas Fiscais: Permite o upload manual de notas fiscais para processamento.**

# **RF008 - Validação de Notas Fiscais: Garante a autenticidade e o status das notas fiscais consultando a SEFAZ.**

# **RF009 - Armazenamento de Notas Fiscais: Armazena notas fiscais de forma organizada para consulta e acesso.**

# **RF010 - Verificação de Inconsistências Fiscais: Realiza conferência de dados fiscais, tributários e sanitários.**

# **RF011 - Notificações de Inconsistências Fiscais: Envia alertas automáticos sobre erros nas NF-e.**

# **RF012 - Geração de Relatórios Fiscais: Permite a geração de relatórios detalhados para análise fiscal e contábil.**

# **Os Requisitos Não Funcionais (RNFs) críticos são:**

# **RNF001 - Performance e Tempo de Resposta: O sistema deve processar notas fiscais e responder eficientemente (máximo 3 segundos para carregamento de página, 5 segundos por nota).**

# **RNF002 - Segurança de Dados: Garante a segurança de dados fiscais e pessoais com criptografia, autenticação robusta (2FA) e controle de acesso baseado em função (RBAC).**

# **RNF004 - Escalabilidade: Deve suportar o aumento do volume de notas fiscais e usuários (até 50.000 NF-e por dia).**

# **RNF005 - Confiabilidade: Alta disponibilidade (99,9% mensal) e backups automáticos.**

# **RNF007 - Processamento em Lote Resiliente: Suporta retry e tolerância a falhas em processos de importação em lote.**

# **RNF008 - Multi-Tenancy e Isolamento de Dados: Garante segregação de dados entre diferentes empresas.**

# **Os requisitos não funcionais, especialmente RNF001 (Performance), RNF004 (Escalabilidade), RNF007 (Processamento em Lote Resiliente) e RNF008 (Multi-Tenancy), são os principais impulsionadores que determinam a escolha da Clean Architecture e do CQRS. Essas exigências impõem a necessidade de um sistema altamente desacoplado, performático e robusto, capaz de lidar com grandes volumes de dados e garantir uma segregação rigorosa. Ao observar a solicitação explícita por Clean Architecture e CQRS, a necessidade de tais padrões arquiteturais é justificada pelas demandas de alta performance (processamento de 50.000 NF-e/dia), escalabilidade e resiliência para operações em lote. Estes são precisamente os desafios que o CQRS e as arquiteturas orientadas a eventos visam resolver. Adicionalmente, o requisito de multi-tenancy (**

# **RNF008) reforça a necessidade de isolamento de dados, o que é beneficiado por uma arquitetura bem-camadada como a Clean Architecture , que impõe limites claros entre as responsabilidades. Consequentemente, os requisitos não funcionais não são apenas complementares, mas são restrições fundamentais que validam a complexidade e os benefícios das soluções arquiteturais propostas. Eles são a causa direta para a adoção dessas abordagens específicas.**

### **1.3. Fundamentos da Clean Architecture e CQRS**

# **A combinação da Clean Architecture e do CQRS fornece uma base robusta para o FreePharma, abordando tanto a complexidade do domínio fiscal quanto as exigências de desempenho.**

# **Clean Architecture: Esta abordagem arquitetural, popularizada por Robert C. Martin (Uncle Bob), enfatiza a separação de preocupações em camadas concêntricas. As camadas internas, que contêm as regras de negócio, são independentes das camadas externas, como interfaces de usuário, bancos de dados ou frameworks. Seus princípios fundamentais incluem a não dependência de módulos de alto nível (lógica de negócio) em módulos de baixo nível (banco de dados, UI), mas sim em abstrações (Princípio da Inversão de Dependência), garantindo a independência das regras de negócio e promovendo a testabilidade. O Spring Boot será utilizado como um framework facilitador, mas não ditará a lógica de negócio central.**

# **CQRS (Command and Query Responsibility Segregation): Este padrão separa as responsabilidades de operações de leitura (Query) e atualização (Command) para um armazenamento de dados. Essa segregação permite a otimização e escalabilidade independentes dos modelos de leitura e escrita, resultando em melhor desempenho e flexibilidade, especialmente em sistemas de grande escala. Frequentemente, o CQRS é combinado com o Event Sourcing para fins de auditabilidade e versionamento.**

# **A decisão de combinar Clean Architecture e CQRS não é meramente um exercício técnico, mas uma escolha estratégica para assegurar a viabilidade a longo prazo, a adaptabilidade e a eficiência operacional do FreePharma em um domínio altamente regulamentado. A Clean Architecture protege a lógica de negócio central de mudanças externas, enquanto o CQRS permite que o sistema lide com o volume de dados inerente e as demandas de desempenho das operações fiscais. Essa combinação suporta diretamente os objetivos de "maior segurança fiscal e eficiência". O sistema FreePharma lida com regras fiscais complexas e em constante evolução, exigindo alta manutenibilidade, e processa grandes volumes de dados, o que demanda desempenho e escalabilidade. Os padrões são complementares: a Clean Architecture assegura que o domínio central (regras fiscais) seja claro e testável, o que é crucial para a conformidade. O CQRS, por sua vez, otimiza os caminhos de execução para entrada de dados (comandos) e saída de dados (consultas), essencial para o processamento de um grande número de NF-e e a geração eficiente de relatórios. Essa sinergia permite que o sistema alcance tanto a robustez em sua lógica central quanto um alto desempenho operacional, traduzindo-se diretamente em valor de negócio ao garantir a conformidade fiscal e a eficiência operacional. Esta é uma decisão arquitetural deliberada e estratégica.**

## **2. Modelagem de Domínio e Diagrama de Classes**

# **Esta seção detalha o modelo de domínio do sistema, apresentando a estrutura inicial das classes, propondo melhorias baseadas em princípios de design e, finalmente, incrementando o diagrama para refletir todos os requisitos identificados.**

### **2.1. Diagrama de Classes Essenciais (Mermaid.js)**

# **A seguir, é apresentado o diagrama de classes fundamental em sintaxe Mermaid.js, conforme fornecido inicialmente. Este diagrama serve como a base para os refinamentos subsequentes, delineando as classes abstratas centrais como EntidadeBase (com id, createdAt, updatedAt) e Pessoa (com id, nome, cpfCnpj, email, telefone), bem como entidades concretas existentes como Farmacia, Unidade, UsuarioAdministrador, Responsavel, Fornecedor, Produto, Estoque, NotaFiscal (e suas especializações NotaFiscalEntrada, NotaFiscalConsumidor), NotaFiscalItem, Inconsistencia, Notificacao e RelatorioFiscal. As relações de herança e associação iniciais definidas no diagrama original estão incluídas.**

# **Code snippet**

# **classDiagram**

# **%% Classes base**

# **class EntidadeBase {**

# **+Long id**

# **+Date createdAt**

# **+Date updatedAt**

# **}**

# **class Pessoa {**

# **+Long id**

# **+String nome**

# **+String cpfCnpj**

# **+String email**

# **+String telefone**

# **}**

# 

# 

# **%% Entidades existentes**

# **class Farmacia {**

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String inscricaoMunicipal**

# **+String endereco**

# **+String telefoneContato**

# **+String emailContato**

# **+String regimeTributario**

# **+CertificadoDigital**

# **certificado**

# **}**

# **class Unidade {**

# **+String tipo**

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String endereco**

# **+String telefone**

# **+String email**

# **+String status**

# **+Date**

# **ultimoAcesso**

# **}**

# **class UsuarioAdministrador {**

# **+String login**

# **+String senha**

# **+Date dataCadastro**

# **+String status**

# **+Date ultimoAcesso**

# **}**

# **class Responsavel {**

# **+String nome**

# **+String cpf**

# **}**

# **class Fornecedor {**

# 

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String endereco**

# **+String email**

# **+String telefone**

# **+String status**

# **+Date dataCadastro**

# **}**

# **class Produto {**

# **+String codigoInterno**

# 

# **+String ean**

# **+String ncm**

# **+String cfop**

# **+List<Aliquota> aliquotas**

# **+Date validade**

# **}**

# **class Estoque {**

# **+Integer quantidadeAtual**

# **+String unidadeMedida**

# **+String motivoAjuste**

# **}**

# **class NotaFiscal {**

# **+String numero**

# 

# **+String chaveAcesso**

# **+String status**

# **+Date dataEmissao**

# **+BigDecimal valorTotal**

# **}**

# **class NotaFiscalEntrada {}**

# **class NotaFiscalConsumidor {}**

# **class NotaFiscalItem {**

# **+Integer quantidade**

# **+BigDecimal valorUnitario**

# **}**

# **class Inconsistencia {**

# **+String tipoErro**

# **+String descricao**

# 

# **}**

# **class Notificacao {**

# **+String tipo**

# **+String mensagem**

# **+Date dataEnvio**

# **}**

# **class RelatorioFiscal {**

# **+String tipoRelatorio**

# **+Date periodoInicio**

# **+Date periodoFim**

# **}**

# 

# **%% Novas classes sugeridas**

# **class ProdutoFornecedor {**

# **+BigDecimal preco**

# 

# **+String codigoFornecedor**

# **}**

# **class ConfiguracaoFiscal {**

# **+String certificadoTipo**

# **+Date certificadoVencimento**

# **}**

# **class LoteProcessamento {**

# **+UUID idLote**

# **+Date dataInicio**

# **+Date dataFim**

# **+String status**

# **}**

# **class HistoricoNotaFiscal {**

# **+Date**

# **dataAlteracao**

# **+String campoAlterado**

# **+String valorAnterior**

# **+String valorNovo**

# **}**

# **class HistoricoValorProduto {**

# **+Date dataAlteracao**

# **+BigDecimal valorAnterior**

# **+BigDecimal valorNovo**

# **}**

# 

# **%% Heranças**

# **EntidadeBase <|-- Farmacia**

# **EntidadeBase <|-- Unidade**

# **EntidadeBase <|-- Fornecedor**

# **EntidadeBase <|-- Produto**

# 

# **EntidadeBase <|-- NotaFiscal**

# **EntidadeBase <|-- Inconsistencia**

# **EntidadeBase <|-- RelatorioFiscal**

# **Pessoa <|-- UsuarioAdministrador**

# **Pessoa <|-- Responsavel**

# **NotaFiscal <|-- NotaFiscalEntrada**

# **NotaFiscal <|-- NotaFiscalConsumidor**

# 

# **%% Associações**

# **Farmacia "1" o-- "\*" Unidade : possui**

# **Farmacia "1" o-- "\*" ConfiguracaoFiscal : configura**

# **Unidade "1" o-- "\*" Estoque : gerencia**

# **Fornecedor "1" o-- "\*" NotaFiscal : emite**

# **Unidade "1" o-- "\*" NotaFiscal : recebe**

# **NotaFiscal "1"**

# **o-- "\*" NotaFiscalItem : contem**

# **NotaFiscalItem "\*" o-- "1" Produto : referencia**

# **Produto "1" o-- "\*" Estoque : estocadoEm**

# **Fornecedor "1" o-- "\*" ProdutoFornecedor : oferta**

# **Produto "1" o-- "\*" ProdutoFornecedor : ofertadoPor**

# **NotaFiscal "1" o-- "\*" Inconsistencia : detecta**

# **Inconsistencia "1" o-- "\*" Notificacao : gera**

# **RelatorioFiscal "..>" NotaFiscal : baseia**

# **LoteProcessamento "1" o-- "\*" NotaFiscal : processa**

# **HistoricoNotaFiscal "1" o-- "\*" NotaFiscal : registra**

# **HistoricoValorProduto "1" o-- "\*"**

# **Produto : registra**

# 

# **Para complementar a clareza do diagrama e facilitar a compreensão, a Tabela 2.1 apresenta um resumo das classes essenciais e suas propriedades críticas, fornecendo uma visão rápida e organizada dos componentes fundamentais do sistema.**

# **Tabela 2.1: Classes Essenciais e Propriedades Críticas (Baseline)**

| **Classe** | **Propriedades Críticas** | **Descrição do Papel no Sistema** |
| --- | --- | --- |
| **EntidadeBase** | **id, createdAt, updatedAt** | **Classe base para auditoria e identificação única de entidades.** |
| **Pessoa** | **id, nome, cpfCnpj, email, telefone** | **Classe base para usuários e responsáveis.** |
| **Farmacia** | **razaoSocial, cnpj, endereco, regimeTributario, certificado** | **Representa a entidade principal do negócio, a farmácia matriz.** |
| **Unidade** | **tipo, cnpj, endereco, status** | **Filiais ou depósitos vinculados à farmácia matriz.** |
| **UsuarioAdministrador** | **login, senha, status** | **Usuário com privilégios de gerenciamento do sistema.** |
| **Responsavel** | **nome, cpf** | **Pessoa responsável por uma unidade ou farmácia.** |
| **Fornecedor** | **razaoSocial, cnpj, endereco, status** | **Entidade que emite as notas fiscais para a farmácia.** |
| **Produto** | **codigoInterno, ean, ncm, cfop, aliquotas, validade** | **Itens comercializados e controlados no estoque.** |
| **Estoque** | **quantidadeAtual, unidadeMedida, motivoAjuste** | **Gerencia as quantidades de produtos em uma unidade.** |
| **NotaFiscal** | **numero, chaveAcesso, status, dataEmissao, valorTotal** | **Documento fiscal base para entrada e saída de produtos.** |
| **NotaFiscalEntrada** | **(Herda de NotaFiscal)** | **Notas fiscais referentes à entrada de produtos.** |
| **NotaFiscalConsumidor** | **(Herda de NotaFiscal)** | **Notas fiscais emitidas diretamente ao consumidor.** |
| **NotaFiscalItem** | **quantidade, valorUnitario** | **Detalhes de cada item dentro de uma nota fiscal.** |
| **Inconsistencia** | **tipoErro, descricao** | **Registro de erros ou divergências detectadas em notas fiscais.** |
| **Notificacao** | **tipo, mensagem, dataEnvio** | **Alertas gerados pelo sistema sobre eventos importantes.** |
| **RelatorioFiscal** | **tipoRelatorio, periodoInicio, periodoFim** | **Documento gerado para análise e conformidade fiscal.** |
| **ProdutoFornecedor** | **preco, codigoFornecedor** | **Associa produtos a fornecedores com preço e código específicos.** |
| **ConfiguracaoFiscal** | **certificadoTipo, certificadoVencimento** | **Configurações fiscais específicas da farmácia.** |
| **LoteProcessamento** | **idLote, dataInicio, dataFim, status** | **Gerencia o processamento em lote de notas fiscais.** |
| **HistoricoNotaFiscal** | **dataAlteracao, campoAlterado, valorAnterior, valorNovo** | **Rastreia alterações em notas fiscais.** |
| **HistoricoValorProduto** | **dataAlteracao, valorAnterior, valorNovo** | **Rastreia alterações de valores de produtos.** |

# **Export to Sheets**

### **2.2. Sugestões de Melhoria para Classes Existentes (SRP, Desacoplamento)**

# **A análise do diagrama de classes inicial revela oportunidades para aplicar princípios de design como o Princípio da Responsabilidade Única (SRP) e o desacoplamento, resultando em um modelo de domínio mais rico e robusto.**

# **A propriedade CertificadoDigital certificado dentro da classe Farmacia indica um acoplamento direto. Para aderir ao SRP e melhorar o desacoplamento,**

# **CertificadoDigital deve ser extraída para uma classe ou Value Object distinto. Esta nova entidade encapsularia suas propriedades específicas, como tipo e vencimento (conforme RF001), e potencialmente o conteúdo binário do certificado (conteudo: byte) e sua senha (senha: String). Isso permite que Farmacia simplesmente mantenha uma referência a esta entidade, tornando o modelo mais modular e flexível. A ausência de CertificadoDigital como uma classe separada no diagrama inicial valida esta adição como uma melhoria necessária.**

# **Similarmente, a propriedade List<Aliquota> aliquotas em Produto sugere uma coleção de taxas.**

# **Aliquota deve ser modelada como um Value Object ou entidade separada, com propriedades como tipoImposto: String, percentual: BigDecimal, vigenciaInicio: Date e vigenciaFim: Date. Isso proporciona um modelo mais granular e extensível para cálculos fiscais complexos e rastreamento de histórico de alíquotas, enriquecendo o domínio.**

# **Embora motivoAjuste seja uma propriedade de Estoque , o requisito**

# **RF006 ("Atualizar estoques em tempo real a partir de NF-e e emitir alertas para níveis mínimos... Permitir ajustes") implica a necessidade de um histórico auditável de movimentações de estoque. Uma entidade dedicada AjusteEstoque modelaria melhor cada transação de ajuste individual, incluindo detalhes como dataAjuste, quantidadeAnterior, quantidadeNova, motivo e o usuarioResponsavel. Isso assegura uma trilha de auditoria completa, essencial para um sistema fiscal. A confirmação de que AjusteEstoque não está no diagrama inicial reforça a necessidade desta adição.**

# **Para Fornecedor, o RF004 lista "Código Interno/EAN", "NCM/CFOP" e "Alíquotas Fiscais" como dados do fornecedor. No entanto, estes são inerentemente atributos do Produto. A classe ProdutoFornecedor, já sugerida em , é crucial aqui. Ela atua como uma classe de associação, representando a relação específica entre um**

# **Fornecedor e um Produto, e contendo atributos únicos a essa relação, como preco: BigDecimal e codigoFornecedor: String. Isso evita a duplicação de dados e esclarece o domínio.**

# **As melhorias propostas elevam o modelo de domínio de uma simples estrutura de dados para um "modelo de domínio rico" que encapsula comportamento e modela explicitamente eventos de negócio (como ajustes de estoque) e relações complexas (como ProdutoFornecedor). Isso é fundamental para um sistema fiscal, onde a auditabilidade, a representação precisa das regras de negócio e a extensibilidade futura são de suma importância. Essas mudanças contribuem diretamente para a criação de um sistema "sólido" e com "padrões de projeto reutilizáveis", conforme a solicitação inicial.**

### **2.3. Incremento do Diagrama de Classes com Requisitos Adicionais**

# **Esta seção detalha as novas classes, propriedades e métodos derivados de uma análise abrangente dos requisitos funcionais e não funcionais e das novas classes sugeridas. Isso forma a base para o diagrama de classes final e refinado em Mermaid.js.**

# **Novas Classes Introduzidas:**

# **CertificadoDigital: Encapsula detalhes do certificado. Propriedades: tipo: String, vencimento: Date, conteudo: byte (dados binários do arquivo do certificado), senha: String. Relação: Farmacia "1" o-- "1" CertificadoDigital : usa.**

# **Aliquota: Representa uma taxa de imposto. Propriedades: tipoImposto: String, percentual: BigDecimal, vigenciaInicio: Date, vigenciaFim: Date. Relação: Produto "1" o-- "\*" Aliquota : possui.**

# **AjusteEstoque: Registra ajustes individuais de estoque. Propriedades: dataAjuste: Date, quantidadeAnterior: Integer, quantidadeNova: Integer, motivo: String. Relação: Estoque "1" o-- "\*" AjusteEstoque : registra.**

# **ConfiguracaoFiscal: (Refinada da sugestão em ) Representa configurações fiscais gerais para uma**

# **Farmacia. Propriedades: regimeTributario: String (movido de Farmacia para melhor SRP), outrasConfiguracoes: Map<String, String> (para extensibilidade). Relação: Farmacia "1" o-- "1" ConfiguracaoFiscal : configura.**

# **HistoricoCompra: Representa um registro de compra de um fornecedor. Propriedades: dataCompra: Date, valorTotal: BigDecimal. Relação: Fornecedor "1" o-- "\*" HistoricoCompra : registra.**

# **Propriedades e Métodos Adicionados/Modificados em Classes Existentes:**

# **EntidadeBase: Adição de idTenant: String. Esta é uma adição crucial para RNF008 (Multi-Tenancy), garantindo a segregação de dados em todas as entidades que herdam dela. A inclusão de idTenant na EntidadeBase é uma decisão arquitetural profunda, não apenas uma coluna de banco de dados. Ela define como os dados são consultados, protegidos e isolados em toda a pilha da aplicação, desde gateways de API até camadas de acesso a dados. Isso exige consideração cuidadosa da propagação do contexto do tenant em todo o sistema.**

# **Farmacia: Adição de status: String (para RF001 "desativação temporária"). Remoção de regimeTributario (movido para ConfiguracaoFiscal), remoção de CertificadoDigital certificado (substituído por relacionamento).**

# **Unidade: Adição de responsavelLocal: Responsavel (para RF002 "Responsável local"). Adição de permissoesAdministrativas: List<String> (para RF002 "Permissões Administrativas").**

# **UsuarioAdministrador: Adição de permissoes: List<String> (para RF003 "Permissões Administrativas").**

# **Fornecedor: Adição de status: String (para RF004).**

# **Produto: Adição de status: String (para gerenciamento geral).**

# **Estoque: Adição de unidadeOrigem: Unidade, unidadeDestino: Unidade (para transferências RF006). Adição do método +ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void (para RF006).**

# **NotaFiscal: Adição de chaveAcesso: String, hashAssinatura: String (para RF008). Adição de valorTotal: BigDecimal (explicitamente de RF009). Adição de farmaciaId: Long, unidadeId: Long (chaves estrangeiras explícitas para o contexto de multi-tenancy). Adição do método abstrato +validar(): List<Inconsistencia>.**

# **NotaFiscalEntrada: Implementação de +validar(): List<Inconsistencia>.**

# **NotaFiscalConsumidor: Implementação de +validar(): List<Inconsistencia>.**

# **Inconsistencia: Adição de dataDeteccao: Date, status: String (ex: 'Aberta', 'Resolvida') para rastreamento (RF010). Relação: Inconsistencia "1" o-- "1" NotaFiscal : referenteA.**

# **Notificacao: Adição de status: String (ex: 'Enviada', 'Lida'), destinatario: UsuarioAdministrador. Relação: Notificacao "1" o-- "1" Inconsistencia : geradaPor.**

# **LoteProcessamento: (Das sugestões em ) Adição de**

# **tipoLote: String (ex: 'Importacao NFe'), progresso: Integer, detalhesErro: String. Relação: LoteProcessamento "1" o-- "\*" NotaFiscal : processa. (Crucial para RNF007).**

# **HistoricoNotaFiscal: (Das sugestões em ) Adição de**

# **usuarioResponsavel: UsuarioAdministrador. Relação: HistoricoNotaFiscal "1" o-- "1" NotaFiscal : registraAlteracaoDe.**

# **HistoricoValorProduto: (Das sugestões em ) Adição de**

# **usuarioResponsavel: UsuarioAdministrador. Relação: HistoricoValorProduto "1" o-- "1" Produto : registraAlteracaoDe.**

# **A adição de idTenant à EntidadeBase (e, portanto, implicitamente à maioria das entidades centrais) impulsionada pelo RNF008 (Multi-Tenancy) representa uma decisão arquitetural fundamental. Não se trata apenas de uma coluna de banco de dados, mas de uma diretriz que molda como os dados são consultados, protegidos e isolados em toda a pilha da aplicação, desde gateways de API até camadas de acesso a dados. Isso exige uma consideração cuidadosa da propagação do contexto do tenant em todo o sistema. Cada consulta ao banco de dados deverá incluir uma cláusula WHERE tenantId = currentTenantId, cada requisição de API deverá carregar informações do tenant (implícita ou explicitamente), e os mecanismos de segurança (RBAC, RNF002) deverão impor controle de acesso ciente do tenant. Esta é uma mudança fundamental no design, não um detalhe menor, e deve ser considerada desde o início para um sistema multi-tenant escalável e seguro.**

# **A interação entre RF006 (atualizações de Estoque a partir de NF-e), RF010 (verificação de Inconsistências), RF011 (Notificações) e RNF007 (Processamento em Lote Resiliente) aponta fortemente para uma arquitetura orientada a eventos. O processamento de uma NF-e (um comando) deve disparar eventos de domínio (por exemplo, NotaFiscalProcessadaEvent, InconsistenciaDetectadaEvent), que podem então ser consumidos assincronamente por vários manipuladores para atualizar o estoque, registrar inconsistências e enviar notificações. Isso se alinha com os benefícios do CQRS e a menção do Kafka em exemplos de CQRS. Essa abordagem assíncrona e orientada a eventos (suportada por Kafka em exemplos de CQRS ) aborda diretamente o**

# **RNF007, permitindo retentativas e tolerância a falhas para etapas individuais. Também melhora a escalabilidade geral do sistema (RNF004) e a capacidade de resposta (RNF001) ao desacoplar essas operações complexas, permitindo que sejam processadas de forma independente e concorrente. Este é um padrão arquitetural crítico para um sistema robusto e de alto throughput.**

# **Diagrama de Classes Refinado (Mermaid.js)**

# **Code snippet**

# **classDiagram**

# **direction LR**

# **%% Base Classes**

# **class EntidadeBase {**

# **+Long id**

# **+Date createdAt**

# **+Date updatedAt**

# **+String idTenant %% RNF008: Multi-Tenancy**

# **}**

# **class Pessoa {**

# **+Long id**

# **+String nome**

# **+String cpfCnpj**

# **+String email**

# **+String telefone**

# **+String idTenant %% RNF008: Multi-Tenancy**

# **}**

# 

# **%% Core Entities (Inherit from EntidadeBase)**

# **EntidadeBase <|-- Farmacia**

# **class Farmacia {**

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String inscricaoMunicipal**

# **+String endereco**

# **+String telefoneContato**

# **+String emailContato**

# **+String status %% RF001: Desativação temporária**

# **}**

# **EntidadeBase <|-- Unidade**

# **class Unidade {**

# **+String tipo**

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String endereco**

# **+String telefone**

# **+String email**

# **+String status**

# **+Date ultimoAcesso**

# **+List~String~ permissoesAdministrativas %% RF002**

# **}**

# **EntidadeBase <|-- Fornecedor**

# **class Fornecedor {**

# **+String razaoSocial**

# **+String nomeFantasia**

# **+String cnpj**

# **+String inscricaoEstadual**

# **+String endereco**

# **+String email**

# **+String telefone**

# **+String status**

# **+Date dataCadastro**

# **}**

# **EntidadeBase <|-- Produto**

# **class Produto {**

# **+String codigoInterno**

# **+String ean**

# **+String ncm**

# **+String cfop**

# **+Date validade**

# **+String status**

# **}**

# **EntidadeBase <|-- NotaFiscal**

# **class NotaFiscal {**

# **+String numero**

# **+String chaveAcesso %% RF008**

# **+String hashAssinatura %% RF008**

# **+String status**

# **+Date dataEmissao**

# **+BigDecimal valorTotal %% RF009**

# **+Long farmaciaId %% Contexto Multi-Tenancy**

# **+Long unidadeId %% Contexto Multi-Tenancy**

# **+abstract validar(): List~Inconsistencia~ %% RF010, Polimorfismo**

# **}**

# **EntidadeBase <|-- Inconsistencia**

# **class Inconsistencia {**

# **+String tipoErro**

# **+String descricao**

# **+Date dataDeteccao %% RF010**

# **+String status %% RF010 (Aberta/Resolvida)**

# **}**

# **EntidadeBase <|-- RelatorioFiscal**

# **class RelatorioFiscal {**

# **+String tipoRelatorio**

# **+Date periodoInicio**

# **+Date periodoFim**

# **}**

# 

# **%% Pessoa Subclasses**

# **Pessoa <|-- UsuarioAdministrador**

# **class UsuarioAdministrador {**

# **+String login**

# **+String senha**

# **+Date dataCadastro**

# **+String status**

# **+Date ultimoAcesso**

# **+List~String~ permissoes %% RF003**

# **}**

# **Pessoa <|-- Responsavel**

# **class Responsavel {**

# **+String nome**

# **+String cpf**

# **}**

# 

# **%% NotaFiscal Subclasses (Polymorphism)**

# **NotaFiscal <|-- NotaFiscalEntrada**

# **class NotaFiscalEntrada {**

# **+validar(): List~Inconsistencia~**

# **}**

# **NotaFiscal <|-- NotaFiscalConsumidor**

# **class NotaFiscalConsumidor {**

# **+validar(): List~Inconsistencia~**

# **}**

# 

# **%% Other Existing Entities**

# **class Estoque {**

# **+Integer quantidadeAtual**

# **+String unidadeMedida**

# **+String motivoAjuste**

# **+ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void %% RF006**

# **}**

# **class NotaFiscalItem {**

# **+Integer quantidade**

# **+BigDecimal valorUnitario**

# **}**

# **class Notificacao {**

# **+String tipo**

# **+String mensagem**

# **+Date dataEnvio**

# **+String status %% RF011 (Enviada/Lida)**

# **}**

# 

# **%% New Classes [1, 1]**

# **class CertificadoDigital {**

# **+String tipo**

# **+Date vencimento**

# **+byte conteudo %% Binary content of the certificate**

# **+String senha**

# **}**

# **class Aliquota {**

# **+String tipoImposto**

# **+BigDecimal percentual**

# **+Date vigenciaInicio**

# **+Date vigenciaFim**

# **}**

# **class AjusteEstoque {**

# **+Date dataAjuste**

# **+Integer quantidadeAnterior**

# **+Integer quantidadeNova**

# **+String motivo**

# **}**

# **class ProdutoFornecedor {**

# **+BigDecimal preco**

# **+String codigoFornecedor**

# **}**

# **class ConfiguracaoFiscal {**

# **+String regimeTributario %% Movido de Farmacia**

# **+Map~String,String~ outrasConfiguracoes %% Para flexibilidade**

# **}**

# **class LoteProcessamento {**

# **+UUID idLote**

# **+Date dataInicio**

# **+Date dataFim**

# **+String status**

# **+String tipoLote %% Ex: 'Importacao NFe'**

# **+Integer progresso**

# **+String detalhesErro**

# **}**

# **class HistoricoNotaFiscal {**

# **+Date dataAlteracao**

# **+String campoAlterado**

# **+String valorAnterior**

# **+String valorNovo**

# **}**

# **class HistoricoValorProduto {**

# **+Date dataAlteracao**

# **+BigDecimal valorAnterior**

# **+BigDecimal valorNovo**

# **}**

# **class HistoricoCompra { %% RF004**

# **+Date dataCompra**

# **+BigDecimal valorTotal**

# **}**

# 

# **%% Associations**

# **Farmacia "1" o-- "\*" Unidade : possui**

# **Farmacia "1" o-- "1" ConfiguracaoFiscal : configura**

# **Farmacia "1" o-- "1" CertificadoDigital : usa**

# **Farmacia "1" o-- "\*" UsuarioAdministrador : gerencia**

# **Unidade "1" o-- "\*" Estoque : gerencia**

# **Unidade "1" o-- "1" Responsavel : temResponsavel**

# **Fornecedor "1" o-- "\*" NotaFiscal : emite**

# **Unidade "1" o-- "\*" NotaFiscal : recebe**

# **NotaFiscal "1" o-- "\*" NotaFiscalItem : contem**

# **NotaFiscalItem "\*" o-- "1" Produto : referencia**

# **Produto "1" o-- "\*" Estoque : estocadoEm**

# **Fornecedor "1" o-- "\*" ProdutoFornecedor : oferta**

# **Produto "1" o-- "\*" ProdutoFornecedor : ofertadoPor**

# **NotaFiscal "1" o-- "\*" Inconsistencia : detecta**

# **Inconsistencia "1" o-- "\*" Notificacao : gera**

# **RelatorioFiscal "..>" NotaFiscal : baseia**

# **LoteProcessamento "1" o-- "\*" NotaFiscal : processa**

# **HistoricoNotaFiscal "1" o-- "1" NotaFiscal : registraAlteracaoDe**

# **HistoricoValorProduto "1" o-- "1" Produto : registraAlteracaoDe**

# **Estoque "1" o-- "\*" AjusteEstoque : registra**

# **AjusteEstoque "\*" o-- "1" UsuarioAdministrador : realizadoPor**

# **Fornecedor "1" o-- "\*" HistoricoCompra : registra**

# **Produto "1" o-- "\*" Aliquota : possui**

# **Notificacao "\*" o-- "1" UsuarioAdministrador : destinadaA**

# 

# **A Tabela 2.2 detalha as propriedades e métodos que foram adicionados ou modificados nas classes, justificando cada alteração com base nos requisitos funcionais (RF) ou não funcionais (RNF) do sistema. Esta tabela proporciona rastreabilidade direta entre os requisitos do sistema e o design detalhado do modelo de domínio, servindo como um registro claro e auditável das decisões de design.**

# **Tabela 2.2: Propriedades e Métodos Adicionados/Modificados por Requisito**

| **Classe** | **Propriedade/Método Adicionado/Modificado** | **Requisito (RF/RNF)** | **Justificativa** |
| --- | --- | --- | --- |
| **EntidadeBase** | **+String idTenant** | **RNF008** | **Essencial para segregação de dados em um ambiente multi-tenant, garantindo que cada farmácia opere com seus dados isolados.** |
| **Farmacia** | **+String status** | **RF001** | **Permite a desativação temporária da farmácia no sistema.** |
| **Farmacia** | **- regimeTributario** | **SRP** | **Movido para ConfiguracaoFiscal para melhor separação de responsabilidades.** |
| **Farmacia** | **- certificado** | **SRP** | **Substituído por associação com a nova classe CertificadoDigital.** |
| **Unidade** | **+Responsavel responsavelLocal** | **RF002** | **Vincula uma unidade a um responsável local, conforme requisito.** |
| **Unidade** | **+List<String> permissoesAdministrativas** | **RF002** | **Define as permissões específicas para a unidade.** |
| **UsuarioAdministrador** | **+List<String> permissoes** | **RF003** | **Detalha as permissões de acesso do usuário administrador.** |
| **Fornecedor** | **+String status** | **RF004** | **Permite controlar o status (ativo/inativo) do fornecedor.** |
| **Produto** | **+String status** | **Geral** | **Adicionado para controle de status do produto (ativo/inativo).** |
| **Estoque** | **+Unidade unidadeOrigem** | **RF006** | **Rastreia a unidade de origem em transferências de estoque.** |
| **Estoque** | **+Unidade unidadeDestino** | **RF006** | **Rastreia a unidade de destino em transferências de estoque.** |
| **Estoque** | **+ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void** | **RF006** | **Encapsula a lógica de ajuste de estoque, permitindo rastreamento e auditoria.** |
| **NotaFiscal** | **+String chaveAcesso** | **RF008** | **Campo obrigatório para validação da NF-e na SEFAZ.** |
| **NotaFiscal** | **+String hashAssinatura** | **RF008** | **Campo para garantir a autenticidade da NF-e.** |
| **NotaFiscal** | **+BigDecimal valorTotal** | **RF009** | **Armazena o valor total da nota fiscal para fácil consulta.** |
| **NotaFiscal** | **+Long farmaciaId** | **RNF008** | **Chave estrangeira para vincular a nota fiscal à farmácia em contexto multi-tenant.** |
| **NotaFiscal** | **+Long unidadeId** | **RNF008** | **Chave estrangeira para vincular a nota fiscal à unidade em contexto multi-tenant.** |
| **NotaFiscal** | **+abstract validar(): List<Inconsistencia>** | **RF010** | **Método polimórfico para validação específica de cada tipo de nota fiscal.** |
| **Inconsistencia** | **+Date dataDeteccao** | **RF010** | **Registra a data em que a inconsistência foi detectada.** |
| **Inconsistencia** | **+String status** | **RF010** | **Permite rastrear o ciclo de vida da inconsistência (e.g., 'Aberta', 'Resolvida').** |
| **Notificacao** | **+String status** | **RF011** | **Rastreia o status da notificação (e.g., 'Enviada', 'Lida').** |
| **Notificacao** | **+UsuarioAdministrador destinatario** | **RF011** | **Define a quem a notificação é destinada.** |
| **LoteProcessamento** | **+String tipoLote** | **RNF007** | **Classifica o tipo de processamento em lote (e.g., 'Importacao NFe').** |
| **LoteProcessamento** | **+Integer progresso** | **RNF007** | **Indica o progresso do processamento do lote.** |
| **LoteProcessamento** | **+String detalhesErro** | **RNF007** | **Armazena detalhes de erros para resiliência e depuração.** |
| **HistoricoNotaFiscal** | **+UsuarioAdministrador usuarioResponsavel** | **RF003** | **Rastreia qual usuário administrador realizou a alteração.** |
| **HistoricoValorProduto** | **+UsuarioAdministrador usuarioResponsavel** | **RF003** | **Rastreia qual usuário administrador realizou a alteração.** |
| **CertificadoDigital** | **Nova Classe** | **RF001, SRP** | **Encapsula detalhes do certificado digital, desacoplando-o da Farmacia.** |
| **Aliquota** | **Nova Classe** | **RF005, SRP** | **Representa alíquotas fiscais de forma granular e extensível.** |
| **AjusteEstoque** | **Nova Classe** | **RF006** | **Modela cada ajuste de estoque como um evento auditável.** |
| **HistoricoCompra** | **Nova Classe** | **RF004** | **Registra o histórico de compras de um fornecedor.** |

# **Export to Sheets**

## **3. Arquitetura do Sistema: Clean Architecture e CQRS em Spring Boot 3**

# **Esta seção aprofunda o blueprint arquitetural do FreePharma, ilustrando as camadas da Clean Architecture e a integração do padrão CQRS dentro do framework Spring Boot 3.**

### **3.1. Diagrama de Arquitetura (Clean Architecture + CQRS)**

# **O diagrama de arquitetura conceitual para o FreePharma ilustra uma estrutura em camadas, aderindo aos princípios da Clean Architecture, com a integração do padrão CQRS. Este diagrama representa o fluxo de comandos e consultas, a segregação dos modelos de leitura e escrita (potencialmente implicando bancos de dados separados ou visões otimizadas), e a interação com os bancos de dados subjacentes. A Regra de Dependência é um ponto central, onde as camadas internas são independentes das camadas externas.**

# **A arquitetura visualizada seria composta por círculos concêntricos: o Core (Domínio) no centro, seguido pela Application (Casos de Uso), Infrastructure (Persistência, Serviços Externos) e a Presentation (API REST) na camada mais externa. As setas de dependência apontariam sempre para dentro, indicando que as camadas externas dependem das internas, mas nunca o contrário. O fluxo de comandos (escritas) e consultas (leituras) seria distinto. Os comandos entrariam pela camada de Apresentação, passariam pela Camada de Aplicação (onde seriam processados por Command Handlers) e interagiriam com o modelo de escrita na Camada de Infraestrutura, que persistiria os dados. As consultas, por sua vez, também entrariam pela Apresentação, seriam processadas por Query Handlers na Camada de Aplicação e acessariam um modelo de leitura otimizado na Camada de Infraestrutura. Eventos de domínio, gerados por comandos bem-sucedidos, seriam publicados e consumidos por componentes na Camada de Infraestrutura para sincronizar o modelo de leitura ou disparar processos assíncronos.**

# **Para garantir a clareza e a interpretação unívoca do diagrama, a Tabela 3.1 define os elementos visuais utilizados. A utilização de uma legenda abrangente elimina a ambiguidade na interpretação do diagrama, assegurando que todos os espectadores, independentemente de seu conhecimento prévio da notação específica, possam compreender corretamente as relações, dependências e fluxos de dados representados. Isso é crucial para uma comunicação técnica eficaz e para a implementação.**

# **Tabela 3.1: Legenda do Diagrama de Arquitetura**

| **Elemento Visual** | **Descrição** |
| --- | --- |
| **Círculo Interno (Verde)** | **Camada Core (Domínio): Regras de Negócio e Entidades.** |
| **Círculo Interno (Azul)** | **Camada Application (Casos de Uso): Lógica de Aplicação, Comandos, Consultas.** |
| **Círculo Intermediário (Amarelo)** | **Camada Infrastructure (Persistência, Serviços Externos): Implementações de Repositórios, Adapters.** |
| **Círculo Externo (Vermelho)** | **Camada Presentation (API REST): Interfaces de Usuário, Controladores.** |
| **Seta Sólida (→)** | **Dependência em tempo de compilação: uma camada depende diretamente da outra.** |
| **Seta Tracejada (⇢)** | **Fluxo de dados ou chamadas em tempo de execução.** |
| **Retângulo (Database)** | **Banco de Dados: Representa o armazenamento de dados.** |
| **Retângulo (External Service)** | **Serviço Externo: Representa sistemas externos (e.g., SEFAZ).** |
| **Retângulo (Message Broker)** | **Broker de Mensagens: Plataforma para comunicação assíncrona (e.g., Kafka).** |
| **Texto "Command Flow"** | **Indica o caminho das operações de escrita.** |
| **Texto "Query Flow"** | **Indica o caminho das operações de leitura.** |
| **Texto "Domain Event"** | **Indica a publicação de eventos de domínio.** |

# **Export to Sheets**

### **3.2. Descrição Detalhada das Camadas**

# **A seguir, uma explicação aprofundada de cada camada da Clean Architecture, detalhando seu propósito, componentes típicos e exemplos concretos relevantes para o sistema FreePharma.**

#### **3.2.1. Camada Core (Domínio)**

# **A Camada Core é o coração do sistema, a camada mais interna e fundamental, completamente independente de quaisquer preocupações externas como frameworks, bancos de dados ou interfaces de usuário. Ela encapsula as regras de negócio essenciais e as entidades, definindo o "o quê" do sistema FreePharma. Sua estabilidade é primordial, pois as mudanças nesta camada devem ser mínimas e impulsionadas exclusivamente pela evolução das regras de negócio.**

# **Os principais componentes desta camada incluem:**

# **Entidades: Objetos de domínio ricos que encapsulam tanto dados quanto comportamento. Exemplos são Farmacia, Produto, NotaFiscal, Unidade, Fornecedor, UsuarioAdministrador, Estoque, Inconsistencia, Notificacao, RelatorioFiscal, LoteProcessamento, HistoricoNotaFiscal, HistoricoValorProduto e HistoricoCompra.**

# **Value Objects: Objetos imutáveis que representam aspectos descritivos do domínio, como CertificadoDigital, Aliquota, Endereco, CpfCnpj e ChaveAcesso.**

# **Domain Services: Operações que envolvem múltiplas entidades ou regras de negócio complexas que não se encaixam naturalmente em uma única entidade. Exemplos incluem ValidadorNotaFiscalDomainService para a lógica de validação abrangente do RF010 e CalculadorImpostoDomainService.**

# **Interfaces (Ports) para Repositórios: Abstrações que definem como os dados são persistidos ou recuperados, como FarmaciaRepositoryPort, ProdutoRepositoryPort e NotaFiscalRepositoryPort. Estes são contratos que a camada de Infraestrutura implementará, aderindo ao Princípio da Inversão de Dependência.**

# **Domain Events: Representações de ocorrências significativas dentro do domínio, como NotaFiscalProcessadaEvent, InconsistenciaDetectadaEvent e EstoqueAtualizadoEvent. Esses eventos são publicados por entidades ou serviços de domínio para sinalizar mudanças de estado.**

# **Por exemplo, o método NotaFiscal.validarInconsistencias() seria um comportamento encapsulado na própria entidade NotaFiscal. Um ValidadorNotaFiscalDomainService.validarMedicamentoControlado(produto, notaFiscalItem) seria um serviço de domínio que coordena a validação de regras mais complexas envolvendo múltiplos objetos.**

# **A Camada Core define o "o quê" do negócio FreePharma, garantindo que as regras fiscais e de gestão de estoque sejam consistentes, imutáveis e protegidas de detalhes externos, independentemente de como são armazenadas ou apresentadas. Dada a natureza complexa e crítica das regras fiscais (RF010), as entidades dentro da Camada Core devem ser "ricas", encapsulando comportamento e lógica de negócio em vez de serem meros contêineres de dados. Um modelo de domínio anêmico dispersaria as regras de negócio por camadas externas, tornando o sistema mais difícil de manter, testar e garantir a conformidade. Por exemplo, a NotaFiscal deve conter a lógica para sua própria validação (validar()), em vez de ser validada apenas por um serviço externo. Isso é vital para um sistema fiscal, pois encapsular a lógica de validação dentro da NotaFiscal garante que a regra de negócio central seja sempre aplicada consistentemente, independentemente do caso de uso, melhorando a robustez e simplificando o teste da lógica fiscal crítica, contribuindo diretamente para o requisito de um sistema "sólido".**

#### **3.2.2. Camada Application (Casos de Uso, Comandos, Consultas)**

# **Esta camada contém as regras de negócio específicas da aplicação e orquestra a execução dos casos de uso. Ela atua como a ponte entre o mundo externo (Camada de Apresentação) e a lógica de negócio central (Camada de Domínio). É aqui que se define o "como" das capacidades da aplicação.**

# **Os principais componentes desta camada incluem:**

# **Use Cases (Interactors): Classes que definem e implementam casos de uso específicos da aplicação, como ImportarNotaFiscalUseCase, CadastrarFarmaciaUseCase e GerarRelatorioFiscalUseCase. Eles orquestram chamadas para entidades de domínio e interfaces de repositório.**

# **Commands: Estruturas de dados imutáveis que representam uma solicitação para alterar o estado do sistema (por exemplo, ImportarNotaFiscalCommand, CadastrarFarmaciaCommand). Elas carregam todos os dados necessários para uma operação de escrita.**

# **Queries: Estruturas de dados imutáveis que representam uma solicitação para recuperar dados sem alterar o estado (por exemplo, BuscarNotasFiscaisQuery, GetFarmaciaByIdQuery). Elas carregam todos os dados necessários para uma operação de leitura.**

# **Command Handlers: Classes responsáveis por processar comandos específicos, orquestrando interações com entidades de domínio e interfaces de persistência.**

# **Query Handlers: Classes responsáveis por processar consultas específicas, recuperando dados principalmente do modelo de leitura ou de fontes de dados otimizadas.**

# **Mediator/Dispatcher: Um mecanismo (por exemplo, utilizando o ApplicationEventPublisher do Spring ou uma implementação personalizada) para despachar comandos e consultas para seus respectivos manipuladores, desacoplando o remetente do receptor.**

# **DTOs (Data Transfer Objects): Modelos de entrada para comandos e consultas (por exemplo, ImportarNotaFiscalRequestDTO) e modelos de saída para respostas (por exemplo, NotaFiscalResponseDTO). Estes são mapeados de/para entidades de domínio.**

# **Um exemplo de fluxo seria um ImportarNotaFiscalCommand sendo recebido por um ImportarNotaFiscalHandler. Este manipulador então utilizaria o NotaFiscalRepositoryPort para salvar a entidade NotaFiscal, dispararia o método NotaFiscal.validar() da camada de domínio e publicaria um InconsistenciaDetectadaEvent se necessário.**

# **A Camada Application orquestra os casos de uso do sistema FreePharma, recebendo comandos e consultas do exterior e coordenando as interações com a Camada Core (para lógica de negócio) e a Camada Infrastructure (para persistência e serviços externos) para executar as funcionalidades de negócio. A adoção explícita do CQRS e a necessidade de uma orquestração clara dos casos de uso implicam fortemente o uso do padrão Mediator dentro da Camada Application. Este padrão desacopla o remetente de um comando ou consulta (por exemplo, um controlador) de seu manipulador específico, aumentando a flexibilidade, a testabilidade e a adesão ao Princípio Aberto/Fechado. Um controlador pode enviar um objeto**

# **Command genérico para um Mediator, que então o despacha para o CommandHandler correto. O controlador não precisa saber qual manipulador processa o comando, apenas que ele envia um comando. Isso promove o Princípio Aberto/Fechado , pois novos comandos e manipuladores podem ser adicionados sem modificar o código do controlador existente. Também simplifica os testes, pois os manipuladores podem ser testados isoladamente. Isso contribui diretamente para um sistema mais "escalável" e "sólido", melhorando a manutenibilidade e reduzindo o impacto das mudanças.**

#### **3.2.3. Camada Infrastructure (Persistência, Serviços Externos)**

# **Esta camada é responsável por implementar as interfaces (ports) definidas nas camadas Core e Application. Ela lida com preocupações externas, como persistência de banco de dados, comunicação com APIs externas, filas de mensagens e armazenamento de arquivos. Esta camada é onde o conhecimento sobre frameworks (Spring Boot, JPA) e sistemas externos reside.**

# **Os principais componentes desta camada incluem:**

# **Implementações de Repositório: Implementações concretas das interfaces de repositório (por exemplo, FarmaciaRepositoryImpl, NotaFiscalRepositoryImpl) usando tecnologias como Spring Data JPA para PostgreSQL.**

# **Adapters de Serviços Externos: Classes que encapsulam chamadas para sistemas externos, como SefazApiAdapter para validação de NF-e (RF008). Estes adaptam as respostas de serviços externos para objetos amigáveis ao domínio.**

# **Implementações de Mensagens: Componentes para publicação e consumo de mensagens, como KafkaEventPublisher e KafkaEventConsumer (como visto em para sincronização CQRS), usados para processamento assíncrono de eventos (por exemplo, notificações**

# **RF011, processamento em lote RNF007).**

# **Processadores em Lote: Serviços ou componentes específicos projetados para lidar com processamento em lote resiliente, como NotaFiscalBatchProcessor para importação de grandes volumes de NF-e (RNF007).**

# **Configurações: Configurações de conexão de banco de dados, endpoints de API externos, detalhes do broker de mensagens.**

# **Exemplos incluem NotaFiscalRepositoryImpl.save(notaFiscalEntity) (persistindo dados), SefazApiAdapter.consultarStatusNFe(chaveAcesso) (chamando API externa) e KafkaEventPublisher.publish(inconsistenciaDetectadaEvent) (enviando mensagem).**

# **A Camada Infrastructure é responsável por conectar o FreePharma ao mundo exterior, implementando os contratos definidos nas camadas internas para persistência de dados, comunicação com serviços externos (SEFAZ) e processamento assíncrono, garantindo que as tecnologias externas sejam apenas "detalhes" para o core da aplicação. Os requisitos para RF008 (consulta à SEFAZ) e RNF007 (Processamento em Lote Resiliente) impõem diretamente a necessidade de tratamento de erros sofisticado, mecanismos de retry e circuit breakers dentro da Camada de Infraestrutura. Dependências externas são inerentemente não confiáveis, e a confiabilidade geral do sistema (RNF005) depende de sua capacidade de lidar graciosamente com falhas nessas interações. Para tanto, a implementação deve prever estratégias como retentativas com *backoff* exponencial, *circuit breakers* para evitar sobrecarga de sistemas externos em falha e *dead-letter queues* para mensagens que não puderam ser processadas após múltiplas tentativas. Isso garante que o sistema possa manter a continuidade operacional mesmo diante de instabilidades de rede ou de serviços externos, contribuindo para a resiliência e a robustez do FreePharma.**

#### **3.2.4. Camada Presentation (API REST)**

# **A camada mais externa, a Camada de Apresentação, é responsável por lidar com a interface do usuário ou com as requisições de outras aplicações. No contexto do FreePharma, que é um sistema de backend com uma API, esta camada será primariamente uma API REST. Ela atua como um adaptador, traduzindo as requisições externas para comandos ou consultas compreendidos pela Camada de Aplicação e formatando as respostas de volta para o formato esperado pelo cliente.**

# **Os principais componentes desta camada incluem:**

# **Controladores (Controllers): Classes que recebem requisições HTTP, validam os dados de entrada e delegam a lógica de negócio para os casos de uso apropriados na Camada de Aplicação. Eles devem ser o mais "finos" possível, focando apenas na coordenação da requisição e resposta.**

# **DTOs (Data Transfer Objects): Modelos de dados específicos para a interface da API, usados para receber dados de entrada (Request DTOs) e para formatar os dados de saída (Response DTOs). Estes DTOs são mapeados de/para os comandos, consultas e modelos de resposta da Camada de Aplicação, garantindo que as entidades de domínio não sejam expostas diretamente à interface externa.**

# **Mappers: Ferramentas ou classes que realizam a conversão entre os DTOs da Camada de Apresentação e os comandos/consultas/modelos de resposta da Camada de Aplicação, bem como as entidades de domínio.**

# **Um exemplo seria um FarmaciaController que recebe uma requisição POST /farmacias com um CadastrarFarmaciaRequestDTO. O controlador validaria este DTO e o mapearia para um CadastrarFarmaciaCommand, que então seria enviado para a Camada de Aplicação para processamento. Após a execução do comando, o resultado seria mapeado para um FarmaciaResponseDTO e retornado ao cliente.**

# **A Camada de Apresentação serve como a porta de entrada para o sistema FreePharma, traduzindo as interações do usuário ou de sistemas externos em operações de negócio e apresentando os resultados de forma compreensível. A separação entre os modelos de API (DTOs) e as entidades de domínio é crucial. Isso permite que a API evolua independentemente da lógica de negócio central, fornecendo controle total sobre o que é exposto aos clientes. Essa separação protege a camada de domínio de mudanças externas, como alterações nos requisitos da API ou na tecnologia de frontend, e assegura que a lógica de negócio permaneça estável e testável. Além disso, ao manter os controladores finos e delegar a lógica de negócio para a Camada de Aplicação, a complexidade é gerenciada de forma eficaz, contribuindo para a manutenibilidade e a escalabilidade do sistema.**

## **Conclusões e Recomendações**

# **A análise detalhada e o design arquitetural do sistema FreePharma, fundamentados na Clean Architecture e no padrão CQRS, demonstram uma abordagem robusta e estratégica para atender aos requisitos complexos de gestão fiscal para farmácias. A escolha desses padrões não é arbitrária, mas uma resposta direta às exigências de performance, escalabilidade, segurança e resiliência, conforme delineado pelos requisitos não funcionais.**

# **A adoção da Clean Architecture garante que as regras de negócio fiscais, que são o cerne do valor do FreePharma, permaneçam isoladas e independentes de tecnologias e frameworks externos. Isso resulta em um sistema intrinsecamente mais testável, manutenível e adaptável a futuras mudanças regulatórias ou de infraestrutura. A ênfase em um modelo de domínio rico, onde as entidades encapsulam comportamentos e validações, é fundamental para assegurar a integridade e a conformidade dos dados fiscais, um aspecto crítico para a operação de farmácias.**

# **A integração do CQRS permite otimizar caminhos de leitura e escrita de forma independente, um benefício crucial para um sistema que precisa processar um alto volume de notas fiscais e gerar relatórios complexos eficientemente. A separação de responsabilidades, juntamente com a implementação de um padrão Mediator na camada de Aplicação, promove um alto grau de desacoplamento, facilitando a adição de novas funcionalidades e a evolução do sistema sem impactar componentes existentes.**

# **A inclusão de um identificador de tenant (idTenant) em todas as entidades base é uma decisão arquitetural que aborda proativamente o requisito de multi-tenancy. Essa abordagem garante a segregação de dados desde a camada de persistência, fundamental para a segurança e confidencialidade das informações de cada farmácia. Além disso, a arquitetura proposta facilita a implementação de fluxos orientados a eventos para processamento assíncrono e resiliente, como a validação de inconsistências e notificações, o que é vital para a confiabilidade do sistema diante de integrações externas e processamento em lote.**

# **Em suma, o design proposto para o FreePharma estabelece uma base tecnológica sólida que não apenas atende aos requisitos funcionais e não funcionais atuais, mas também posiciona o sistema para crescimento e adaptação contínuos em um ambiente regulatório e de mercado dinâmico. A aderência a princípios de design robustos e a padrões arquiteturais comprovados assegura que o FreePharma será uma ferramenta eficiente, segura e confiável para a gestão fiscal de farmácias.**

# **latório Técnico: Arquitetura e Modelagem de Domínio para o Sistema FreePharma**

Este relatório detalha a arquitetura e a modelagem de domínio para o sistema FreePharma, com foco na implementação de uma arquitetura limpa (Clean Architecture) e no padrão CQRS (Command Query Responsibility Segregation) utilizando Spring Boot 3. O objetivo é fornecer um plano técnico robusto que garanta a escalabilidade, solidez e manutenibilidade do sistema, alinhado aos requisitos fiscais e operacionais de farmácias.

## **1. Visão Geral do Sistema FreePharma e Contexto Arquitetural**

Esta seção estabelece o entendimento fundamental do sistema FreePharma, seu propósito central, os requisitos que o impulsionam e as escolhas arquiteturais que guiarão seu desenvolvimento.

### **1.1. Propósito e Escopo do FreePharma**

O sistema FreePharma é concebido como uma solução abrangente de gestão fiscal, especificamente adaptada para farmácias. Seu objetivo primordial é automatizar a organização, leitura e análise de notas fiscais eletrônicas (NF-e e NFC-e), visando aprimorar a segurança fiscal e otimizar os processos de escrituração e declaração de obrigações tributárias, como Imposto de Renda (IR), SPED Fiscal e Sintegra.

As funcionalidades essenciais do FreePharma incluem a importação e gestão centralizada de notas fiscais, a detecção automática de inconsistências fiscais – como divergências de valores, erros de classificação tributária (NCM, CFOP, CST) e duplicidades – e a geração de relatórios inteligentes. Estes relatórios são projetados para evidenciar irregularidades, identificar oportunidades de correção e auxiliar na tomada de decisões contábeis e fiscais. A natureza do sistema, focada em "segurança fiscal" e "eficiência no processo de escrituração e declaração de obrigações tributárias", ressalta que a integridade, precisão e conformidade dos dados não são meras funcionalidades, mas sim requisitos fundamentais e inegociáveis. Isso implica que o design do sistema deve priorizar mecanismos robustos de validação, auditabilidade e tratamento de erros para prevenir repercussões legais e financeiras para as farmácias. A escolha do PostgreSQL como sistema de gerenciamento de banco de dados relacional apoia inerentemente a integridade transacional necessária para um sistema fiscal.

### **1.2. Requisitos Funcionais e Não Funcionais Chave**

Os requisitos do sistema FreePharma, tanto funcionais quanto não funcionais, exercem uma influência direta e significativa sobre as decisões de design arquitetural.

Os principais Requisitos Funcionais (RFs) incluem:

* RF001 - Cadastro de Farmácia: Permite registrar informações essenciais da farmácia, incluindo dados fiscais e certificado digital.
* RF002 - Cadastro de Unidades (Filiais, Depósitos): Gerencia unidades vinculadas à matriz, com controle de estoque e obrigações específicas.
* RF003 - Cadastro de Usuário Administrador: Permite criar e gerenciar usuários com permissões de acesso.
* RF004 - Gerenciamento de Fornecedores: Habilita o cadastro e a gestão de fornecedores, facilitando a associação de notas fiscais.
* RF005 - Gerenciamento de Produtos: Permite o cadastro e a gestão de produtos, incluindo dados fiscais e de validade.
* RF006 - Gerenciamento de Estoque: Atualiza estoques em tempo real a partir de NF-e e emite alertas.
* RF007 - Importação de Notas Fiscais: Permite o upload manual de notas fiscais para processamento.
* RF008 - Validação de Notas Fiscais: Garante a autenticidade e o status das notas fiscais consultando a SEFAZ.
* RF009 - Armazenamento de Notas Fiscais: Armazena notas fiscais de forma organizada para consulta e acesso.
* RF010 - Verificação de Inconsistências Fiscais: Realiza conferência de dados fiscais, tributários e sanitários.
* RF011 - Notificações de Inconsistências Fiscais: Envia alertas automáticos sobre erros nas NF-e.
* RF012 - Geração de Relatórios Fiscais: Permite a geração de relatórios detalhados para análise fiscal e contábil.

Os Requisitos Não Funcionais (RNFs) críticos são:

* RNF001 - Performance e Tempo de Resposta: O sistema deve processar notas fiscais e responder eficientemente (máximo 3 segundos para carregamento de página, 5 segundos por nota).
* RNF002 - Segurança de Dados: Garante a segurança de dados fiscais e pessoais com criptografia, autenticação robusta (2FA) e controle de acesso baseado em função (RBAC).
* RNF004 - Escalabilidade: Deve suportar o aumento do volume de notas fiscais e usuários (até 50.000 NF-e por dia).
* RNF005 - Confiabilidade: Alta disponibilidade (99,9% mensal) e backups automáticos.
* RNF007 - Processamento em Lote Resiliente: Suporta retry e tolerância a falhas em processos de importação em lote.
* RNF008 - Multi-Tenancy e Isolamento de Dados: Garante segregação de dados entre diferentes empresas.

Os requisitos não funcionais, especialmente RNF001 (Performance), RNF004 (Escalabilidade), RNF007 (Processamento em Lote Resiliente) e RNF008 (Multi-Tenancy), são os principais impulsionadores que determinam a escolha da Clean Architecture e do CQRS. Essas exigências impõem a necessidade de um sistema altamente desacoplado, performático e robusto, capaz de lidar com grandes volumes de dados e garantir uma segregação rigorosa. Ao observar a solicitação explícita por Clean Architecture e CQRS, a necessidade de tais padrões arquiteturais é justificada pelas demandas de alta performance (processamento de 50.000 NF-e/dia), escalabilidade e resiliência para operações em lote. Estes são precisamente os desafios que o CQRS e as arquiteturas orientadas a eventos visam resolver. Adicionalmente, o requisito de multi-tenancy (

RNF008) reforça a necessidade de isolamento de dados, o que é beneficiado por uma arquitetura bem-camadada como a Clean Architecture , que impõe limites claros entre as responsabilidades. Consequentemente, os requisitos não funcionais não são apenas complementares, mas são restrições fundamentais que validam a complexidade e os benefícios das soluções arquiteturais propostas. Eles são a causa direta para a adoção dessas abordagens específicas.

### **1.3. Fundamentos da Clean Architecture e CQRS**

A combinação da Clean Architecture e do CQRS fornece uma base robusta para o FreePharma, abordando tanto a complexidade do domínio fiscal quanto as exigências de desempenho.

**Clean Architecture:** Esta abordagem arquitetural, popularizada por Robert C. Martin (Uncle Bob), enfatiza a separação de preocupações em camadas concêntricas. As camadas internas, que contêm as regras de negócio, são independentes das camadas externas, como interfaces de usuário, bancos de dados ou frameworks. Seus princípios fundamentais incluem a não dependência de módulos de alto nível (lógica de negócio) em módulos de baixo nível (banco de dados, UI), mas sim em abstrações (Princípio da Inversão de Dependência), garantindo a independência das regras de negócio e promovendo a testabilidade. O Spring Boot será utilizado como um framework facilitador, mas não ditará a lógica de negócio central.

**CQRS (Command and Query Responsibility Segregation):** Este padrão separa as responsabilidades de operações de leitura (Query) e atualização (Command) para um armazenamento de dados. Essa segregação permite a otimização e escalabilidade independentes dos modelos de leitura e escrita, resultando em melhor desempenho e flexibilidade, especialmente em sistemas de grande escala. Frequentemente, o CQRS é combinado com o Event Sourcing para fins de auditabilidade e versionamento.

A decisão de combinar Clean Architecture e CQRS não é meramente um exercício técnico, mas uma escolha estratégica para assegurar a viabilidade a longo prazo, a adaptabilidade e a eficiência operacional do FreePharma em um domínio altamente regulamentado. A Clean Architecture protege a lógica de negócio central de mudanças externas, enquanto o CQRS permite que o sistema lide com o volume de dados inerente e as demandas de desempenho das operações fiscais. Essa combinação suporta diretamente os objetivos de "maior segurança fiscal e eficiência". O sistema FreePharma lida com regras fiscais complexas e em constante evolução, exigindo alta manutenibilidade, e processa grandes volumes de dados, o que demanda desempenho e escalabilidade. Os padrões são complementares: a Clean Architecture assegura que o domínio central (regras fiscais) seja claro e testável, o que é crucial para a conformidade. O CQRS, por sua vez, otimiza os caminhos de execução para entrada de dados (comandos) e saída de dados (consultas), essencial para o processamento de um grande número de NF-e e a geração eficiente de relatórios. Essa sinergia permite que o sistema alcance tanto a robustez em sua lógica central quanto um alto desempenho operacional, traduzindo-se diretamente em valor de negócio ao garantir a conformidade fiscal e a eficiência operacional. Esta é uma decisão arquitetural deliberada e estratégica.

## **2. Modelagem de Domínio e Diagrama de Classes**

Esta seção detalha o modelo de domínio do sistema, apresentando a estrutura inicial das classes, propondo melhorias baseadas em princípios de design e, finalmente, incrementando o diagrama para refletir todos os requisitos identificados.

### **2.1. Diagrama de Classes Essenciais (Mermaid.js)**

A seguir, é apresentado o diagrama de classes fundamental em sintaxe Mermaid.js, conforme fornecido inicialmente. Este diagrama serve como a base para os refinamentos subsequentes, delineando as classes abstratas centrais como EntidadeBase (com id, createdAt, updatedAt) e Pessoa (com id, nome, cpfCnpj, email, telefone), bem como entidades concretas existentes como Farmacia, Unidade, UsuarioAdministrador, Responsavel, Fornecedor, Produto, Estoque, NotaFiscal (e suas especializações NotaFiscalEntrada, NotaFiscalConsumidor), NotaFiscalItem, Inconsistencia, Notificacao e RelatorioFiscal. As relações de herança e associação iniciais definidas no diagrama original estão incluídas.

Code snippet

classDiagram

%% Classes base

class EntidadeBase {

+Long id

+Date createdAt

+Date updatedAt

}

class Pessoa {

+Long id

+String nome

+String cpfCnpj

+String email

+String telefone

}

%% Entidades existentes

class Farmacia {

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String inscricaoMunicipal

+String endereco

+String telefoneContato

+String emailContato

+String regimeTributario

+CertificadoDigital

certificado

}

class Unidade {

+String tipo

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String endereco

+String telefone

+String email

+String status

+Date

ultimoAcesso

}

class UsuarioAdministrador {

+String login

+String senha

+Date dataCadastro

+String status

+Date ultimoAcesso

}

class Responsavel {

+String nome

+String cpf

}

class Fornecedor {

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String endereco

+String email

+String telefone

+String status

+Date dataCadastro

}

class Produto {

+String codigoInterno

+String ean

+String ncm

+String cfop

+List<Aliquota> aliquotas

+Date validade

}

class Estoque {

+Integer quantidadeAtual

+String unidadeMedida

+String motivoAjuste

}

class NotaFiscal {

+String numero

+String chaveAcesso

+String status

+Date dataEmissao

+BigDecimal valorTotal

}

class NotaFiscalEntrada {}

class NotaFiscalConsumidor {}

class NotaFiscalItem {

+Integer quantidade

+BigDecimal valorUnitario

}

class Inconsistencia {

+String tipoErro

+String descricao

}

class Notificacao {

+String tipo

+String mensagem

+Date dataEnvio

}

class RelatorioFiscal {

+String tipoRelatorio

+Date periodoInicio

+Date periodoFim

}

%% Novas classes sugeridas

class ProdutoFornecedor {

+BigDecimal preco

+String codigoFornecedor

}

class ConfiguracaoFiscal {

+String certificadoTipo

+Date certificadoVencimento

}

class LoteProcessamento {

+UUID idLote

+Date dataInicio

+Date dataFim

+String status

}

class HistoricoNotaFiscal {

+Date

dataAlteracao

+String campoAlterado

+String valorAnterior

+String valorNovo

}

class HistoricoValorProduto {

+Date dataAlteracao

+BigDecimal valorAnterior

+BigDecimal valorNovo

}

%% Heranças

EntidadeBase <|-- Farmacia

EntidadeBase <|-- Unidade

EntidadeBase <|-- Fornecedor

EntidadeBase <|-- Produto

EntidadeBase <|-- NotaFiscal

EntidadeBase <|-- Inconsistencia

EntidadeBase <|-- RelatorioFiscal

Pessoa <|-- UsuarioAdministrador

Pessoa <|-- Responsavel

NotaFiscal <|-- NotaFiscalEntrada

NotaFiscal <|-- NotaFiscalConsumidor

%% Associações

Farmacia "1" o-- "\*" Unidade : possui

Farmacia "1" o-- "\*" ConfiguracaoFiscal : configura

Unidade "1" o-- "\*" Estoque : gerencia

Fornecedor "1" o-- "\*" NotaFiscal : emite

Unidade "1" o-- "\*" NotaFiscal : recebe

NotaFiscal "1"

o-- "\*" NotaFiscalItem : contem

NotaFiscalItem "\*" o-- "1" Produto : referencia

Produto "1" o-- "\*" Estoque : estocadoEm

Fornecedor "1" o-- "\*" ProdutoFornecedor : oferta

Produto "1" o-- "\*" ProdutoFornecedor : ofertadoPor

NotaFiscal "1" o-- "\*" Inconsistencia : detecta

Inconsistencia "1" o-- "\*" Notificacao : gera

RelatorioFiscal "..>" NotaFiscal : baseia

LoteProcessamento "1" o-- "\*" NotaFiscal : processa

HistoricoNotaFiscal "1" o-- "\*" NotaFiscal : registra

HistoricoValorProduto "1" o-- "\*"

Produto : registra

Para complementar a clareza do diagrama e facilitar a compreensão, a Tabela 2.1 apresenta um resumo das classes essenciais e suas propriedades críticas, fornecendo uma visão rápida e organizada dos componentes fundamentais do sistema.

**Tabela 2.1: Classes Essenciais e Propriedades Críticas (Baseline)**

| Classe | Propriedades Críticas | Descrição do Papel no Sistema |
| --- | --- | --- |
| EntidadeBase | id, createdAt, updatedAt | Classe base para auditoria e identificação única de entidades. |
| Pessoa | id, nome, cpfCnpj, email, telefone | Classe base para usuários e responsáveis. |
| Farmacia | razaoSocial, cnpj, endereco, regimeTributario, certificado | Representa a entidade principal do negócio, a farmácia matriz. |
| Unidade | tipo, cnpj, endereco, status | Filiais ou depósitos vinculados à farmácia matriz. |
| UsuarioAdministrador | login, senha, status | Usuário com privilégios de gerenciamento do sistema. |
| Responsavel | nome, cpf | Pessoa responsável por uma unidade ou farmácia. |
| Fornecedor | razaoSocial, cnpj, endereco, status | Entidade que emite as notas fiscais para a farmácia. |
| Produto | codigoInterno, ean, ncm, cfop, aliquotas, validade | Itens comercializados e controlados no estoque. |
| Estoque | quantidadeAtual, unidadeMedida, motivoAjuste | Gerencia as quantidades de produtos em uma unidade. |
| NotaFiscal | numero, chaveAcesso, status, dataEmissao, valorTotal | Documento fiscal base para entrada e saída de produtos. |
| NotaFiscalEntrada | (Herda de NotaFiscal) | Notas fiscais referentes à entrada de produtos. |
| NotaFiscalConsumidor | (Herda de NotaFiscal) | Notas fiscais emitidas diretamente ao consumidor. |
| NotaFiscalItem | quantidade, valorUnitario | Detalhes de cada item dentro de uma nota fiscal. |
| Inconsistencia | tipoErro, descricao | Registro de erros ou divergências detectadas em notas fiscais. |
| Notificacao | tipo, mensagem, dataEnvio | Alertas gerados pelo sistema sobre eventos importantes. |
| RelatorioFiscal | tipoRelatorio, periodoInicio, periodoFim | Documento gerado para análise e conformidade fiscal. |
| ProdutoFornecedor | preco, codigoFornecedor | Associa produtos a fornecedores com preço e código específicos. |
| ConfiguracaoFiscal | certificadoTipo, certificadoVencimento | Configurações fiscais específicas da farmácia. |
| LoteProcessamento | idLote, dataInicio, dataFim, status | Gerencia o processamento em lote de notas fiscais. |
| HistoricoNotaFiscal | dataAlteracao, campoAlterado, valorAnterior, valorNovo | Rastreia alterações em notas fiscais. |
| HistoricoValorProduto | dataAlteracao, valorAnterior, valorNovo | Rastreia alterações de valores de produtos. |

Export to Sheets

### **2.2. Sugestões de Melhoria para Classes Existentes (SRP, Desacoplamento)**

A análise do diagrama de classes inicial revela oportunidades para aplicar princípios de design como o Princípio da Responsabilidade Única (SRP) e o desacoplamento, resultando em um modelo de domínio mais rico e robusto.

A propriedade CertificadoDigital certificado dentro da classe Farmacia indica um acoplamento direto. Para aderir ao SRP e melhorar o desacoplamento,

CertificadoDigital deve ser extraída para uma classe ou Value Object distinto. Esta nova entidade encapsularia suas propriedades específicas, como tipo e vencimento (conforme RF001), e potencialmente o conteúdo binário do certificado (conteudo: byte) e sua senha (senha: String). Isso permite que Farmacia simplesmente mantenha uma referência a esta entidade, tornando o modelo mais modular e flexível. A ausência de CertificadoDigital como uma classe separada no diagrama inicial valida esta adição como uma melhoria necessária.

Similarmente, a propriedade List<Aliquota> aliquotas em Produto sugere uma coleção de taxas.

Aliquota deve ser modelada como um Value Object ou entidade separada, com propriedades como tipoImposto: String, percentual: BigDecimal, vigenciaInicio: Date e vigenciaFim: Date. Isso proporciona um modelo mais granular e extensível para cálculos fiscais complexos e rastreamento de histórico de alíquotas, enriquecendo o domínio.

Embora motivoAjuste seja uma propriedade de Estoque , o requisito

RF006 ("Atualizar estoques em tempo real a partir de NF-e e emitir alertas para níveis mínimos... Permitir ajustes") implica a necessidade de um histórico auditável de movimentações de estoque. Uma entidade dedicada AjusteEstoque modelaria melhor cada transação de ajuste individual, incluindo detalhes como dataAjuste, quantidadeAnterior, quantidadeNova, motivo e o usuarioResponsavel. Isso assegura uma trilha de auditoria completa, essencial para um sistema fiscal. A confirmação de que AjusteEstoque não está no diagrama inicial reforça a necessidade desta adição.

Para Fornecedor, o RF004 lista "Código Interno/EAN", "NCM/CFOP" e "Alíquotas Fiscais" como dados do fornecedor. No entanto, estes são inerentemente atributos do Produto. A classe ProdutoFornecedor, já sugerida em , é crucial aqui. Ela atua como uma classe de associação, representando a relação específica entre um

Fornecedor e um Produto, e contendo atributos únicos a essa relação, como preco: BigDecimal e codigoFornecedor: String. Isso evita a duplicação de dados e esclarece o domínio.

As melhorias propostas elevam o modelo de domínio de uma simples estrutura de dados para um "modelo de domínio rico" que encapsula comportamento e modela explicitamente eventos de negócio (como ajustes de estoque) e relações complexas (como ProdutoFornecedor). Isso é fundamental para um sistema fiscal, onde a auditabilidade, a representação precisa das regras de negócio e a extensibilidade futura são de suma importância. Essas mudanças contribuem diretamente para a criação de um sistema "sólido" e com "padrões de projeto reutilizáveis", conforme a solicitação inicial.

### **2.3. Incremento do Diagrama de Classes com Requisitos Adicionais**

Esta seção detalha as novas classes, propriedades e métodos derivados de uma análise abrangente dos requisitos funcionais e não funcionais e das novas classes sugeridas. Isso forma a base para o diagrama de classes final e refinado em Mermaid.js.

**Novas Classes Introduzidas:**

* CertificadoDigital: Encapsula detalhes do certificado. Propriedades: tipo: String, vencimento: Date, conteudo: byte (dados binários do arquivo do certificado), senha: String. Relação: Farmacia "1" o-- "1" CertificadoDigital : usa.
* Aliquota: Representa uma taxa de imposto. Propriedades: tipoImposto: String, percentual: BigDecimal, vigenciaInicio: Date, vigenciaFim: Date. Relação: Produto "1" o-- "\*" Aliquota : possui.
* AjusteEstoque: Registra ajustes individuais de estoque. Propriedades: dataAjuste: Date, quantidadeAnterior: Integer, quantidadeNova: Integer, motivo: String. Relação: Estoque "1" o-- "\*" AjusteEstoque : registra.
* ConfiguracaoFiscal: (Refinada da sugestão em ) Representa configurações fiscais gerais para uma
* Farmacia. Propriedades: regimeTributario: String (movido de Farmacia para melhor SRP), outrasConfiguracoes: Map<String, String> (para extensibilidade). Relação: Farmacia "1" o-- "1" ConfiguracaoFiscal : configura.
* HistoricoCompra: Representa um registro de compra de um fornecedor. Propriedades: dataCompra: Date, valorTotal: BigDecimal. Relação: Fornecedor "1" o-- "\*" HistoricoCompra : registra.

**Propriedades e Métodos Adicionados/Modificados em Classes Existentes:**

* EntidadeBase: Adição de idTenant: String. Esta é uma adição crucial para RNF008 (Multi-Tenancy), garantindo a segregação de dados em todas as entidades que herdam dela. A inclusão de idTenant na EntidadeBase é uma decisão arquitetural profunda, não apenas uma coluna de banco de dados. Ela define como os dados são consultados, protegidos e isolados em toda a pilha da aplicação, desde gateways de API até camadas de acesso a dados. Isso exige consideração cuidadosa da propagação do contexto do tenant em todo o sistema.
* Farmacia: Adição de status: String (para RF001 "desativação temporária"). Remoção de regimeTributario (movido para ConfiguracaoFiscal), remoção de CertificadoDigital certificado (substituído por relacionamento).
* Unidade: Adição de responsavelLocal: Responsavel (para RF002 "Responsável local"). Adição de permissoesAdministrativas: List<String> (para RF002 "Permissões Administrativas").
* UsuarioAdministrador: Adição de permissoes: List<String> (para RF003 "Permissões Administrativas").
* Fornecedor: Adição de status: String (para RF004).
* Produto: Adição de status: String (para gerenciamento geral).
* Estoque: Adição de unidadeOrigem: Unidade, unidadeDestino: Unidade (para transferências RF006). Adição do método +ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void (para RF006).
* NotaFiscal: Adição de chaveAcesso: String, hashAssinatura: String (para RF008). Adição de valorTotal: BigDecimal (explicitamente de RF009). Adição de farmaciaId: Long, unidadeId: Long (chaves estrangeiras explícitas para o contexto de multi-tenancy). Adição do método abstrato +validar(): List<Inconsistencia>.
* NotaFiscalEntrada: Implementação de +validar(): List<Inconsistencia>.
* NotaFiscalConsumidor: Implementação de +validar(): List<Inconsistencia>.
* Inconsistencia: Adição de dataDeteccao: Date, status: String (ex: 'Aberta', 'Resolvida') para rastreamento (RF010). Relação: Inconsistencia "1" o-- "1" NotaFiscal : referenteA.
* Notificacao: Adição de status: String (ex: 'Enviada', 'Lida'), destinatario: UsuarioAdministrador. Relação: Notificacao "1" o-- "1" Inconsistencia : geradaPor.
* LoteProcessamento: (Das sugestões em ) Adição de
* tipoLote: String (ex: 'Importacao NFe'), progresso: Integer, detalhesErro: String. Relação: LoteProcessamento "1" o-- "\*" NotaFiscal : processa. (Crucial para RNF007).
* HistoricoNotaFiscal: (Das sugestões em ) Adição de
* usuarioResponsavel: UsuarioAdministrador. Relação: HistoricoNotaFiscal "1" o-- "1" NotaFiscal : registraAlteracaoDe.
* HistoricoValorProduto: (Das sugestões em ) Adição de
* usuarioResponsavel: UsuarioAdministrador. Relação: HistoricoValorProduto "1" o-- "1" Produto : registraAlteracaoDe.

A adição de idTenant à EntidadeBase (e, portanto, implicitamente à maioria das entidades centrais) impulsionada pelo RNF008 (Multi-Tenancy) representa uma decisão arquitetural fundamental. Não se trata apenas de uma coluna de banco de dados, mas de uma diretriz que molda como os dados são consultados, protegidos e isolados em toda a pilha da aplicação, desde gateways de API até camadas de acesso a dados. Isso exige uma consideração cuidadosa da propagação do contexto do tenant em todo o sistema. Cada consulta ao banco de dados deverá incluir uma cláusula WHERE tenantId = currentTenantId, cada requisição de API deverá carregar informações do tenant (implícita ou explicitamente), e os mecanismos de segurança (RBAC, RNF002) deverão impor controle de acesso ciente do tenant. Esta é uma mudança fundamental no design, não um detalhe menor, e deve ser considerada desde o início para um sistema multi-tenant escalável e seguro.

A interação entre RF006 (atualizações de Estoque a partir de NF-e), RF010 (verificação de Inconsistências), RF011 (Notificações) e RNF007 (Processamento em Lote Resiliente) aponta fortemente para uma arquitetura orientada a eventos. O processamento de uma NF-e (um comando) deve disparar eventos de domínio (por exemplo, NotaFiscalProcessadaEvent, InconsistenciaDetectadaEvent), que podem então ser consumidos assincronamente por vários manipuladores para atualizar o estoque, registrar inconsistências e enviar notificações. Isso se alinha com os benefícios do CQRS e a menção do Kafka em exemplos de CQRS. Essa abordagem assíncrona e orientada a eventos (suportada por Kafka em exemplos de CQRS ) aborda diretamente o

RNF007, permitindo retentativas e tolerância a falhas para etapas individuais. Também melhora a escalabilidade geral do sistema (RNF004) e a capacidade de resposta (RNF001) ao desacoplar essas operações complexas, permitindo que sejam processadas de forma independente e concorrente. Este é um padrão arquitetural crítico para um sistema robusto e de alto throughput.

**Diagrama de Classes Refinado (Mermaid.js)**

Code snippet

classDiagram

direction LR

%% Base Classes

class EntidadeBase {

+Long id

+Date createdAt

+Date updatedAt

+String idTenant %% RNF008: Multi-Tenancy

}

class Pessoa {

+Long id

+String nome

+String cpfCnpj

+String email

+String telefone

+String idTenant %% RNF008: Multi-Tenancy

}

%% Core Entities (Inherit from EntidadeBase)

EntidadeBase <|-- Farmacia

class Farmacia {

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String inscricaoMunicipal

+String endereco

+String telefoneContato

+String emailContato

+String status %% RF001: Desativação temporária

}

EntidadeBase <|-- Unidade

class Unidade {

+String tipo

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String endereco

+String telefone

+String email

+String status

+Date ultimoAcesso

+List~String~ permissoesAdministrativas %% RF002

}

EntidadeBase <|-- Fornecedor

class Fornecedor {

+String razaoSocial

+String nomeFantasia

+String cnpj

+String inscricaoEstadual

+String endereco

+String email

+String telefone

+String status

+Date dataCadastro

}

EntidadeBase <|-- Produto

class Produto {

+String codigoInterno

+String ean

+String ncm

+String cfop

+Date validade

+String status

}

EntidadeBase <|-- NotaFiscal

class NotaFiscal {

+String numero

+String chaveAcesso %% RF008

+String hashAssinatura %% RF008

+String status

+Date dataEmissao

+BigDecimal valorTotal %% RF009

+Long farmaciaId %% Contexto Multi-Tenancy

+Long unidadeId %% Contexto Multi-Tenancy

+abstract validar(): List~Inconsistencia~ %% RF010, Polimorfismo

}

EntidadeBase <|-- Inconsistencia

class Inconsistencia {

+String tipoErro

+String descricao

+Date dataDeteccao %% RF010

+String status %% RF010 (Aberta/Resolvida)

}

EntidadeBase <|-- RelatorioFiscal

class RelatorioFiscal {

+String tipoRelatorio

+Date periodoInicio

+Date periodoFim

}

%% Pessoa Subclasses

Pessoa <|-- UsuarioAdministrador

class UsuarioAdministrador {

+String login

+String senha

+Date dataCadastro

+String status

+Date ultimoAcesso

+List~String~ permissoes %% RF003

}

Pessoa <|-- Responsavel

class Responsavel {

+String nome

+String cpf

}

%% NotaFiscal Subclasses (Polymorphism)

NotaFiscal <|-- NotaFiscalEntrada

class NotaFiscalEntrada {

+validar(): List~Inconsistencia~

}

NotaFiscal <|-- NotaFiscalConsumidor

class NotaFiscalConsumidor {

+validar(): List~Inconsistencia~

}

%% Other Existing Entities

class Estoque {

+Integer quantidadeAtual

+String unidadeMedida

+String motivoAjuste

+ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void %% RF006

}

class NotaFiscalItem {

+Integer quantidade

+BigDecimal valorUnitario

}

class Notificacao {

+String tipo

+String mensagem

+Date dataEnvio

+String status %% RF011 (Enviada/Lida)

}

%% New Classes [1, 1]

class CertificadoDigital {

+String tipo

+Date vencimento

+byte conteudo %% Binary content of the certificate

+String senha

}

class Aliquota {

+String tipoImposto

+BigDecimal percentual

+Date vigenciaInicio

+Date vigenciaFim

}

class AjusteEstoque {

+Date dataAjuste

+Integer quantidadeAnterior

+Integer quantidadeNova

+String motivo

}

class ProdutoFornecedor {

+BigDecimal preco

+String codigoFornecedor

}

class ConfiguracaoFiscal {

+String regimeTributario %% Movido de Farmacia

+Map~String,String~ outrasConfiguracoes %% Para flexibilidade

}

class LoteProcessamento {

+UUID idLote

+Date dataInicio

+Date dataFim

+String status

+String tipoLote %% Ex: 'Importacao NFe'

+Integer progresso

+String detalhesErro

}

class HistoricoNotaFiscal {

+Date dataAlteracao

+String campoAlterado

+String valorAnterior

+String valorNovo

}

class HistoricoValorProduto {

+Date dataAlteracao

+BigDecimal valorAnterior

+BigDecimal valorNovo

}

class HistoricoCompra { %% RF004

+Date dataCompra

+BigDecimal valorTotal

}

%% Associations

Farmacia "1" o-- "\*" Unidade : possui

Farmacia "1" o-- "1" ConfiguracaoFiscal : configura

Farmacia "1" o-- "1" CertificadoDigital : usa

Farmacia "1" o-- "\*" UsuarioAdministrador : gerencia

Unidade "1" o-- "\*" Estoque : gerencia

Unidade "1" o-- "1" Responsavel : temResponsavel

Fornecedor "1" o-- "\*" NotaFiscal : emite

Unidade "1" o-- "\*" NotaFiscal : recebe

NotaFiscal "1" o-- "\*" NotaFiscalItem : contem

NotaFiscalItem "\*" o-- "1" Produto : referencia

Produto "1" o-- "\*" Estoque : estocadoEm

Fornecedor "1" o-- "\*" ProdutoFornecedor : oferta

Produto "1" o-- "\*" ProdutoFornecedor : ofertadoPor

NotaFiscal "1" o-- "\*" Inconsistencia : detecta

Inconsistencia "1" o-- "\*" Notificacao : gera

RelatorioFiscal "..>" NotaFiscal : baseia

LoteProcessamento "1" o-- "\*" NotaFiscal : processa

HistoricoNotaFiscal "1" o-- "1" NotaFiscal : registraAlteracaoDe

HistoricoValorProduto "1" o-- "1" Produto : registraAlteracaoDe

Estoque "1" o-- "\*" AjusteEstoque : registra

AjusteEstoque "\*" o-- "1" UsuarioAdministrador : realizadoPor

Fornecedor "1" o-- "\*" HistoricoCompra : registra

Produto "1" o-- "\*" Aliquota : possui

Notificacao "\*" o-- "1" UsuarioAdministrador : destinadaA

A Tabela 2.2 detalha as propriedades e métodos que foram adicionados ou modificados nas classes, justificando cada alteração com base nos requisitos funcionais (RF) ou não funcionais (RNF) do sistema. Esta tabela proporciona rastreabilidade direta entre os requisitos do sistema e o design detalhado do modelo de domínio, servindo como um registro claro e auditável das decisões de design.

**Tabela 2.2: Propriedades e Métodos Adicionados/Modificados por Requisito**

| Classe | Propriedade/Método Adicionado/Modificado | Requisito (RF/RNF) | Justificativa |
| --- | --- | --- | --- |
| EntidadeBase | +String idTenant | RNF008 | Essencial para segregação de dados em um ambiente multi-tenant, garantindo que cada farmácia opere com seus dados isolados. |
| Farmacia | +String status | RF001 | Permite a desativação temporária da farmácia no sistema. |
| Farmacia | - regimeTributario | SRP | Movido para ConfiguracaoFiscal para melhor separação de responsabilidades. |
| Farmacia | - certificado | SRP | Substituído por associação com a nova classe CertificadoDigital. |
| Unidade | +Responsavel responsavelLocal | RF002 | Vincula uma unidade a um responsável local, conforme requisito. |
| Unidade | +List<String> permissoesAdministrativas | RF002 | Define as permissões específicas para a unidade. |
| UsuarioAdministrador | +List<String> permissoes | RF003 | Detalha as permissões de acesso do usuário administrador. |
| Fornecedor | +String status | RF004 | Permite controlar o status (ativo/inativo) do fornecedor. |
| Produto | +String status | Geral | Adicionado para controle de status do produto (ativo/inativo). |
| Estoque | +Unidade unidadeOrigem | RF006 | Rastreia a unidade de origem em transferências de estoque. |
| Estoque | +Unidade unidadeDestino | RF006 | Rastreia a unidade de destino em transferências de estoque. |
| Estoque | +ajustarQuantidade(quantidade: Integer, motivo: String, usuario: UsuarioAdministrador): void | RF006 | Encapsula a lógica de ajuste de estoque, permitindo rastreamento e auditoria. |
| NotaFiscal | +String chaveAcesso | RF008 | Campo obrigatório para validação da NF-e na SEFAZ. |
| NotaFiscal | +String hashAssinatura | RF008 | Campo para garantir a autenticidade da NF-e. |
| NotaFiscal | +BigDecimal valorTotal | RF009 | Armazena o valor total da nota fiscal para fácil consulta. |
| NotaFiscal | +Long farmaciaId | RNF008 | Chave estrangeira para vincular a nota fiscal à farmácia em contexto multi-tenant. |
| NotaFiscal | +Long unidadeId | RNF008 | Chave estrangeira para vincular a nota fiscal à unidade em contexto multi-tenant. |
| NotaFiscal | +abstract validar(): List<Inconsistencia> | RF010 | Método polimórfico para validação específica de cada tipo de nota fiscal. |
| Inconsistencia | +Date dataDeteccao | RF010 | Registra a data em que a inconsistência foi detectada. |
| Inconsistencia | +String status | RF010 | Permite rastrear o ciclo de vida da inconsistência (e.g., 'Aberta', 'Resolvida'). |
| Notificacao | +String status | RF011 | Rastreia o status da notificação (e.g., 'Enviada', 'Lida'). |
| Notificacao | +UsuarioAdministrador destinatario | RF011 | Define a quem a notificação é destinada. |
| LoteProcessamento | +String tipoLote | RNF007 | Classifica o tipo de processamento em lote (e.g., 'Importacao NFe'). |
| LoteProcessamento | +Integer progresso | RNF007 | Indica o progresso do processamento do lote. |
| LoteProcessamento | +String detalhesErro | RNF007 | Armazena detalhes de erros para resiliência e depuração. |
| HistoricoNotaFiscal | +UsuarioAdministrador usuarioResponsavel | RF003 | Rastreia qual usuário administrador realizou a alteração. |
| HistoricoValorProduto | +UsuarioAdministrador usuarioResponsavel | RF003 | Rastreia qual usuário administrador realizou a alteração. |
| CertificadoDigital | Nova Classe | RF001, SRP | Encapsula detalhes do certificado digital, desacoplando-o da Farmacia. |
| Aliquota | Nova Classe | RF005, SRP | Representa alíquotas fiscais de forma granular e extensível. |
| AjusteEstoque | Nova Classe | RF006 | Modela cada ajuste de estoque como um evento auditável. |
| HistoricoCompra | Nova Classe | RF004 | Registra o histórico de compras de um fornecedor. |

Export to Sheets

## **3. Arquitetura do Sistema: Clean Architecture e CQRS em Spring Boot 3**

Esta seção aprofunda o blueprint arquitetural do FreePharma, ilustrando as camadas da Clean Architecture e a integração do padrão CQRS dentro do framework Spring Boot 3.

### **3.1. Diagrama de Arquitetura (Clean Architecture + CQRS)**

O diagrama de arquitetura conceitual para o FreePharma ilustra uma estrutura em camadas, aderindo aos princípios da Clean Architecture, com a integração do padrão CQRS. Este diagrama representa o fluxo de comandos e consultas, a segregação dos modelos de leitura e escrita (potencialmente implicando bancos de dados separados ou visões otimizadas), e a interação com os bancos de dados subjacentes. A Regra de Dependência é um ponto central, onde as camadas internas são independentes das camadas externas.

A arquitetura visualizada seria composta por círculos concêntricos: o **Core (Domínio)** no centro, seguido pela **Application (Casos de Uso)**, **Infrastructure (Persistência, Serviços Externos)** e a **Presentation (API REST)** na camada mais externa. As setas de dependência apontariam sempre para dentro, indicando que as camadas externas dependem das internas, mas nunca o contrário. O fluxo de comandos (escritas) e consultas (leituras) seria distinto. Os comandos entrariam pela camada de Apresentação, passariam pela Camada de Aplicação (onde seriam processados por Command Handlers) e interagiriam com o modelo de escrita na Camada de Infraestrutura, que persistiria os dados. As consultas, por sua vez, também entrariam pela Apresentação, seriam processadas por Query Handlers na Camada de Aplicação e acessariam um modelo de leitura otimizado na Camada de Infraestrutura. Eventos de domínio, gerados por comandos bem-sucedidos, seriam publicados e consumidos por componentes na Camada de Infraestrutura para sincronizar o modelo de leitura ou disparar processos assíncronos.

Para garantir a clareza e a interpretação unívoca do diagrama, a Tabela 3.1 define os elementos visuais utilizados. A utilização de uma legenda abrangente elimina a ambiguidade na interpretação do diagrama, assegurando que todos os espectadores, independentemente de seu conhecimento prévio da notação específica, possam compreender corretamente as relações, dependências e fluxos de dados representados. Isso é crucial para uma comunicação técnica eficaz e para a implementação.

**Tabela 3.1: Legenda do Diagrama de Arquitetura**

| Elemento Visual | Descrição |
| --- | --- |
| Círculo Interno (Verde) | Camada Core (Domínio): Regras de Negócio e Entidades. |
| Círculo Interno (Azul) | Camada Application (Casos de Uso): Lógica de Aplicação, Comandos, Consultas. |
| Círculo Intermediário (Amarelo) | Camada Infrastructure (Persistência, Serviços Externos): Implementações de Repositórios, Adapters. |
| Círculo Externo (Vermelho) | Camada Presentation (API REST): Interfaces de Usuário, Controladores. |
| Seta Sólida (→) | Dependência em tempo de compilação: uma camada depende diretamente da outra. |
| Seta Tracejada (⇢) | Fluxo de dados ou chamadas em tempo de execução. |
| Retângulo (Database) | Banco de Dados: Representa o armazenamento de dados. |
| Retângulo (External Service) | Serviço Externo: Representa sistemas externos (e.g., SEFAZ). |
| Retângulo (Message Broker) | Broker de Mensagens: Plataforma para comunicação assíncrona (e.g., Kafka). |
| Texto "Command Flow" | Indica o caminho das operações de escrita. |
| Texto "Query Flow" | Indica o caminho das operações de leitura. |
| Texto "Domain Event" | Indica a publicação de eventos de domínio. |

Export to Sheets

### **3.2. Descrição Detalhada das Camadas**

A seguir, uma explicação aprofundada de cada camada da Clean Architecture, detalhando seu propósito, componentes típicos e exemplos concretos relevantes para o sistema FreePharma.

#### **3.2.1. Camada Core (Domínio)**

A Camada Core é o coração do sistema, a camada mais interna e fundamental, completamente independente de quaisquer preocupações externas como frameworks, bancos de dados ou interfaces de usuário. Ela encapsula as regras de negócio essenciais e as entidades, definindo o "o quê" do sistema FreePharma. Sua estabilidade é primordial, pois as mudanças nesta camada devem ser mínimas e impulsionadas exclusivamente pela evolução das regras de negócio.

Os principais **componentes** desta camada incluem:

* **Entidades:** Objetos de domínio ricos que encapsulam tanto dados quanto comportamento. Exemplos são Farmacia, Produto, NotaFiscal, Unidade, Fornecedor, UsuarioAdministrador, Estoque, Inconsistencia, Notificacao, RelatorioFiscal, LoteProcessamento, HistoricoNotaFiscal, HistoricoValorProduto e HistoricoCompra.
* **Value Objects:** Objetos imutáveis que representam aspectos descritivos do domínio, como CertificadoDigital, Aliquota, Endereco, CpfCnpj e ChaveAcesso.
* **Domain Services:** Operações que envolvem múltiplas entidades ou regras de negócio complexas que não se encaixam naturalmente em uma única entidade. Exemplos incluem ValidadorNotaFiscalDomainService para a lógica de validação abrangente do RF010 e CalculadorImpostoDomainService.
* **Interfaces (Ports) para Repositórios:** Abstrações que definem como os dados são persistidos ou recuperados, como FarmaciaRepositoryPort, ProdutoRepositoryPort e NotaFiscalRepositoryPort. Estes são contratos que a camada de Infraestrutura implementará, aderindo ao Princípio da Inversão de Dependência.
* **Domain Events:** Representações de ocorrências significativas dentro do domínio, como NotaFiscalProcessadaEvent, InconsistenciaDetectadaEvent e EstoqueAtualizadoEvent. Esses eventos são publicados por entidades ou serviços de domínio para sinalizar mudanças de estado.

Por exemplo, o método NotaFiscal.validarInconsistencias() seria um comportamento encapsulado na própria entidade NotaFiscal. Um ValidadorNotaFiscalDomainService.validarMedicamentoControlado(produto, notaFiscalItem) seria um serviço de domínio que coordena a validação de regras mais complexas envolvendo múltiplos objetos.

A Camada Core define o "o quê" do negócio FreePharma, garantindo que as regras fiscais e de gestão de estoque sejam consistentes, imutáveis e protegidas de detalhes externos, independentemente de como são armazenadas ou apresentadas. Dada a natureza complexa e crítica das regras fiscais (RF010), as entidades dentro da Camada Core devem ser "ricas", encapsulando comportamento e lógica de negócio em vez de serem meros contêineres de dados. Um modelo de domínio anêmico dispersaria as regras de negócio por camadas externas, tornando o sistema mais difícil de manter, testar e garantir a conformidade. Por exemplo, a NotaFiscal deve conter a lógica para sua própria validação (validar()), em vez de ser validada apenas por um serviço externo. Isso é vital para um sistema fiscal, pois encapsular a lógica de validação dentro da NotaFiscal garante que a regra de negócio central seja sempre aplicada consistentemente, independentemente do caso de uso, melhorando a robustez e simplificando o teste da lógica fiscal crítica, contribuindo diretamente para o requisito de um sistema "sólido".

#### **3.2.2. Camada Application (Casos de Uso, Comandos, Consultas)**

Esta camada contém as regras de negócio específicas da aplicação e orquestra a execução dos casos de uso. Ela atua como a ponte entre o mundo externo (Camada de Apresentação) e a lógica de negócio central (Camada de Domínio). É aqui que se define o "como" das capacidades da aplicação.

Os principais **componentes** desta camada incluem:

* **Use Cases (Interactors):** Classes que definem e implementam casos de uso específicos da aplicação, como ImportarNotaFiscalUseCase, CadastrarFarmaciaUseCase e GerarRelatorioFiscalUseCase. Eles orquestram chamadas para entidades de domínio e interfaces de repositório.
* **Commands:** Estruturas de dados imutáveis que representam uma solicitação para alterar o estado do sistema (por exemplo, ImportarNotaFiscalCommand, CadastrarFarmaciaCommand). Elas carregam todos os dados necessários para uma operação de escrita.
* **Queries:** Estruturas de dados imutáveis que representam uma solicitação para recuperar dados sem alterar o estado (por exemplo, BuscarNotasFiscaisQuery, GetFarmaciaByIdQuery). Elas carregam todos os dados necessários para uma operação de leitura.
* **Command Handlers:** Classes responsáveis por processar comandos específicos, orquestrando interações com entidades de domínio e interfaces de persistência.
* **Query Handlers:** Classes responsáveis por processar consultas específicas, recuperando dados principalmente do modelo de leitura ou de fontes de dados otimizadas.
* **Mediator/Dispatcher:** Um mecanismo (por exemplo, utilizando o ApplicationEventPublisher do Spring ou uma implementação personalizada) para despachar comandos e consultas para seus respectivos manipuladores, desacoplando o remetente do receptor.
* **DTOs (Data Transfer Objects):** Modelos de entrada para comandos e consultas (por exemplo, ImportarNotaFiscalRequestDTO) e modelos de saída para respostas (por exemplo, NotaFiscalResponseDTO). Estes são mapeados de/para entidades de domínio.

Um exemplo de fluxo seria um ImportarNotaFiscalCommand sendo recebido por um ImportarNotaFiscalHandler. Este manipulador então utilizaria o NotaFiscalRepositoryPort para salvar a entidade NotaFiscal, dispararia o método NotaFiscal.validar() da camada de domínio e publicaria um InconsistenciaDetectadaEvent se necessário.

A Camada Application orquestra os casos de uso do sistema FreePharma, recebendo comandos e consultas do exterior e coordenando as interações com a Camada Core (para lógica de negócio) e a Camada Infrastructure (para persistência e serviços externos) para executar as funcionalidades de negócio. A adoção explícita do CQRS e a necessidade de uma orquestração clara dos casos de uso implicam fortemente o uso do padrão Mediator dentro da Camada Application. Este padrão desacopla o remetente de um comando ou consulta (por exemplo, um controlador) de seu manipulador específico, aumentando a flexibilidade, a testabilidade e a adesão ao Princípio Aberto/Fechado. Um controlador pode enviar um objeto

Command genérico para um Mediator, que então o despacha para o CommandHandler correto. O controlador não precisa saber qual manipulador processa o comando, apenas que ele envia um comando. Isso promove o Princípio Aberto/Fechado , pois novos comandos e manipuladores podem ser adicionados sem modificar o código do controlador existente. Também simplifica os testes, pois os manipuladores podem ser testados isoladamente. Isso contribui diretamente para um sistema mais "escalável" e "sólido", melhorando a manutenibilidade e reduzindo o impacto das mudanças.

#### **3.2.3. Camada Infrastructure (Persistência, Serviços Externos)**

Esta camada é responsável por implementar as interfaces (ports) definidas nas camadas Core e Application. Ela lida com preocupações externas, como persistência de banco de dados, comunicação com APIs externas, filas de mensagens e armazenamento de arquivos. Esta camada é onde o conhecimento sobre frameworks (Spring Boot, JPA) e sistemas externos reside.

Os principais **componentes** desta camada incluem:

* **Implementações de Repositório:** Implementações concretas das interfaces de repositório (por exemplo, FarmaciaRepositoryImpl, NotaFiscalRepositoryImpl) usando tecnologias como Spring Data JPA para PostgreSQL.
* **Adapters de Serviços Externos:** Classes que encapsulam chamadas para sistemas externos, como SefazApiAdapter para validação de NF-e (RF008). Estes adaptam as respostas de serviços externos para objetos amigáveis ao domínio.
* **Implementações de Mensagens:** Componentes para publicação e consumo de mensagens, como KafkaEventPublisher e KafkaEventConsumer (como visto em para sincronização CQRS), usados para processamento assíncrono de eventos (por exemplo, notificações
* RF011, processamento em lote RNF007).
* **Processadores em Lote:** Serviços ou componentes específicos projetados para lidar com processamento em lote resiliente, como NotaFiscalBatchProcessor para importação de grandes volumes de NF-e (RNF007).
* **Configurações:** Configurações de conexão de banco de dados, endpoints de API externos, detalhes do broker de mensagens.

Exemplos incluem NotaFiscalRepositoryImpl.save(notaFiscalEntity) (persistindo dados), SefazApiAdapter.consultarStatusNFe(chaveAcesso) (chamando API externa) e KafkaEventPublisher.publish(inconsistenciaDetectadaEvent) (enviando mensagem).

A Camada Infrastructure é responsável por conectar o FreePharma ao mundo exterior, implementando os contratos definidos nas camadas internas para persistência de dados, comunicação com serviços externos (SEFAZ) e processamento assíncrono, garantindo que as tecnologias externas sejam apenas "detalhes" para o core da aplicação. Os requisitos para RF008 (consulta à SEFAZ) e RNF007 (Processamento em Lote Resiliente) impõem diretamente a necessidade de tratamento de erros sofisticado, mecanismos de retry e circuit breakers dentro da Camada de Infraestrutura. Dependências externas são inerentemente não confiáveis, e a confiabilidade geral do sistema (RNF005) depende de sua capacidade de lidar graciosamente com falhas nessas interações. Para tanto, a implementação deve prever estratégias como retentativas com *backoff* exponencial, *circuit breakers* para evitar sobrecarga de sistemas externos em falha e *dead-letter queues* para mensagens que não puderam ser processadas após múltiplas tentativas. Isso garante que o sistema possa manter a continuidade operacional mesmo diante de instabilidades de rede ou de serviços externos, contribuindo para a resiliência e a robustez do FreePharma.

#### **3.2.4. Camada Presentation (API REST)**

A camada mais externa, a Camada de Apresentação, é responsável por lidar com a interface do usuário ou com as requisições de outras aplicações. No contexto do FreePharma, que é um sistema de backend com uma API, esta camada será primariamente uma API REST. Ela atua como um adaptador, traduzindo as requisições externas para comandos ou consultas compreendidos pela Camada de Aplicação e formatando as respostas de volta para o formato esperado pelo cliente.

Os principais **componentes** desta camada incluem:

* **Controladores (Controllers):** Classes que recebem requisições HTTP, validam os dados de entrada e delegam a lógica de negócio para os casos de uso apropriados na Camada de Aplicação. Eles devem ser o mais "finos" possível, focando apenas na coordenação da requisição e resposta.
* **DTOs (Data Transfer Objects):** Modelos de dados específicos para a interface da API, usados para receber dados de entrada (Request DTOs) e para formatar os dados de saída (Response DTOs). Estes DTOs são mapeados de/para os comandos, consultas e modelos de resposta da Camada de Aplicação, garantindo que as entidades de domínio não sejam expostas diretamente à interface externa.
* **Mappers:** Ferramentas ou classes que realizam a conversão entre os DTOs da Camada de Apresentação e os comandos/consultas/modelos de resposta da Camada de Aplicação, bem como as entidades de domínio.

Um exemplo seria um FarmaciaController que recebe uma requisição POST /farmacias com um CadastrarFarmaciaRequestDTO. O controlador validaria este DTO e o mapearia para um CadastrarFarmaciaCommand, que então seria enviado para a Camada de Aplicação para processamento. Após a execução do comando, o resultado seria mapeado para um FarmaciaResponseDTO e retornado ao cliente.

A Camada de Apresentação serve como a porta de entrada para o sistema FreePharma, traduzindo as interações do usuário ou de sistemas externos em operações de negócio e apresentando os resultados de forma compreensível. A separação entre os modelos de API (DTOs) e as entidades de domínio é crucial. Isso permite que a API evolua independentemente da lógica de negócio central, fornecendo controle total sobre o que é exposto aos clientes. Essa separação protege a camada de domínio de mudanças externas, como alterações nos requisitos da API ou na tecnologia de frontend, e assegura que a lógica de negócio permaneça estável e testável. Além disso, ao manter os controladores finos e delegar a lógica de negócio para a Camada de Aplicação, a complexidade é gerenciada de forma eficaz, contribuindo para a manutenibilidade e a escalabilidade do sistema.

## **Conclusões e Recomendações**

A análise detalhada e o design arquitetural do sistema FreePharma, fundamentados na Clean Architecture e no padrão CQRS, demonstram uma abordagem robusta e estratégica para atender aos requisitos complexos de gestão fiscal para farmácias. A escolha desses padrões não é arbitrária, mas uma resposta direta às exigências de performance, escalabilidade, segurança e resiliência, conforme delineado pelos requisitos não funcionais.

A adoção da Clean Architecture garante que as regras de negócio fiscais, que são o cerne do valor do FreePharma, permaneçam isoladas e independentes de tecnologias e frameworks externos. Isso resulta em um sistema intrinsecamente mais testável, manutenível e adaptável a futuras mudanças regulatórias ou de infraestrutura. A ênfase em um modelo de domínio rico, onde as entidades encapsulam comportamentos e validações, é fundamental para assegurar a integridade e a conformidade dos dados fiscais, um aspecto crítico para a operação de farmácias.

A integração do CQRS permite otimizar caminhos de leitura e escrita de forma independente, um benefício crucial para um sistema que precisa processar um alto volume de notas fiscais e gerar relatórios complexos eficientemente. A separação de responsabilidades, juntamente com a implementação de um padrão Mediator na camada de Aplicação, promove um alto grau de desacoplamento, facilitando a adição de novas funcionalidades e a evolução do sistema sem impactar componentes existentes.

A inclusão de um identificador de tenant (idTenant) em todas as entidades base é uma decisão arquitetural que aborda proativamente o requisito de multi-tenancy. Essa abordagem garante a segregação de dados desde a camada de persistência, fundamental para a segurança e confidencialidade das informações de cada farmácia. Além disso, a arquitetura proposta facilita a implementação de fluxos orientados a eventos para processamento assíncrono e resiliente, como a validação de inconsistências e notificações, o que é vital para a confiabilidade do sistema diante de integrações externas e processamento em lote.

Em suma, o design proposto para o FreePharma estabelece uma base tecnológica sólida que não apenas atende aos requisitos funcionais e não funcionais atuais, mas também posiciona o sistema para crescimento e adaptação contínuos em um ambiente regulatório e de mercado dinâmico. A aderência a princípios de design robustos e a padrões arquiteturais comprovados assegura que o FreePharma será uma ferramenta eficiente, segura e confiável para a gestão fiscal de farmácias.