UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Leonardo Saponi de Souza

Evolução Incremental do Sistema exomaDB: Modernização Tecnológica e Implementação de Módulo de Visualização de Anotações Genômicas

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Leonardo Saponi de Souza

Evolução Incremental do Sistema exomaDB: Modernização Tecnológica e Implementação de Módulo de Visualização de Anotações Genômicas

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Anderson Rodrigues dos Santos

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Leonardo Saponi de Souza

Evolução Incremental do Sistema exomaDB: Modernização Tecnológica e Implementação de Módulo de Visualização de Anotações Genômicas

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 26 de setembro de 2025:

Prof. Dr. Anderson Rodrigues dos Santos
Orientador

Dr. Alexsandro Santos Soares
Professor

Dr. Rafael Dias Araújo
Professor

Uberlândia, Brasil
2025

Resumo

As doenças raras representam um desafio de saúde pública global, afetando aproximadamente 300 milhões de pessoas mundialmente. A análise genômica, especialmente por meio de tecnologias de sequenciamento de nova geração (NGS), tornou-se fundamental para o diagnóstico dessas condições. O Laboratório de Genética (LABGEN) da Universidade Federal de Uberlândia utiliza o sistema exomaDB para gerenciar dados genômicos em colaboração com o Centro de Adição e Saúde Mental (CAMH). Embora funcional, o sistema possui limitações técnicas que comprometem sua evolução e manutenção. Este trabalho teve como objetivo modernizar a base tecnológica do exomaDB e implementar um módulo de visualização de anotações genômicas através de uma estratégia de evolução incremental. A metodologia incluiu diagnóstico da arquitetura original, migração para Maven, e desenvolvimento de nova arquitetura em camadas baseada nos princípios SOLID. Os resultados demonstraram melhoria notável em manutenibilidade (redução de 85% no tempo de implementação de filtros), testabilidade (cobertura de 85% nas camadas críticas) e performance (redução de 80% no tempo de carregamento através de *lazy loading*). A nova arquitetura foi validada como modelo de referência para a evolução sustentável do sistema, contribuindo diretamente para a pesquisa em doenças raras.

Palavras-chave: exomaDB. Doenças raras. Genômica. Arquitetura de software. Bioinformática.

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application Programming Interface
CAMH	Centro de Adição e Saúde Mental
DIP	Dependency Inversion Principle
ISP	Interface Segregation Principle
JPA	Java Persistence API
JSF	JavaServer Faces
LABGEN	Laboratório de Genética
LSP	Liskov Substitution Principle
MVC	Model-View-Controller
NGS	Next-Generation Sequencing
OCP	Open-Closed Principle
ORM	Object-Relational Mapping
POJO	Plain Old Java Object
SRP	Single Responsibility Principle
UFU	Universidade Federal de Uberlândia
VCF	Variant Call Format

Sumário

1 Introdução

As doenças raras, apesar da baixa prevalência individual, representam um desafio de saúde pública global, afetando um número estimado em 300 milhões de pessoas em todo o mundo (WAKAP et al., 2020). Cerca de 80% dessas condições têm origem genética, o que torna a análise genômica uma peça-chave para o diagnóstico (BRASIL. MINISTÉRIO DA SAÚDE, 2014). Nesse contexto, um diagnóstico molecular preciso é decisivo não apenas para definir tratamentos adequados, mas, fundamentalmente, para proporcionar uma melhoria real na qualidade de vida dos pacientes (BOYCOTT et al., 2017).

A investigação de doenças raras foi transformada pela genômica aplicada, que utiliza tecnologias de sequenciamento de nova geração (NGS), como o sequenciamento do exoma e do genoma completo, para identificar as variantes genéticas que causam essas enfermidades (BIESECKER; GREEN, 2014; WRIGHT; FITZPATRICK; FIRTH, 2018). Graças ao rápido avanço tecnológico e à redução de custos, a análise de DNA tornou-se mais acessível, integrando os testes genéticos à prática clínica e abrindo caminho para tratamentos personalizados (HUANG et al., 2020; DOHERTY; BAMSHAD; NICKERSON, 2020).

É neste cenário que o Laboratório de Genética (LABGEN) do Instituto de Genética e Bioquímica (INGEB) da Universidade Federal de Uberlândia (UFU) desenvolve seu projeto de pesquisa para rastrear variantes genéticas, em colaboração com o Centro de Adição e Saúde Mental (CAMH).

Para gerenciar o grande volume de dados genômicos gerados por essa pesquisa, o LABGEN utiliza o sistema exomaDB. A plataforma foi desenvolvida em parceria com o Laboratório de Biologia Computacional e Bioinformática (Comp2Bio) da UFU, sob a coordenação do Prof. Dr. Anderson Rodrigues dos Santos. Embora o sistema tenha atendido bem às necessidades iniciais do projeto, ele hoje apresenta limitações técnicas que comprometem sua evolução e a sua manutenção a longo prazo.

Afinal, o ciclo de vida de um software não termina após a implementação; a evolução contínua é essencial para que ele se mantenha relevante e útil (SOMMERVILLE, 2011). À medida que a pesquisa avança e as necessidades dos usuários mudam, novas demandas surgem, exigindo adaptações na estrutura do sistema (LEHMAN, 1980). No exomaDB, dois obstáculos críticos impedem essa evolução: primeiro, um gerenciamento manual de dependências que torna as atualizações arriscadas e complexas; segundo, a falta de padrões de arquitetura consistentes, que resultou em um código pouco modular e difícil de expandir.

Diante desses desafios, uma abordagem de evolução incremental mostra-se mais

prudente e eficaz do que uma refatoração completa do sistema (SOMMERVILLE, 2011; FOWLER, 1999). Por isso, este trabalho foi planejado para atacar as limitações do exomaDB em duas frentes complementares. A primeira foi modernizar a base tecnológica do projeto, migrando o gerenciamento de dependências para o Maven.

A segunda, motivada por uma demanda direta dos pesquisadores, foi implementar um novo módulo de visualização de anotações genômicas e, por fim, apresentar uma análise das melhorias obtidas, validando a hipótese de que a modernização arquitetural gerou ganhos práticos e demonstráveis de manutenibilidade, testabilidade e performance.

1.1 Objetivos

Motivado pela necessidade de superar as limitações técnicas do sistema exomaDB e de apoiar mais eficazmente a pesquisa em doenças raras no LABGEN, o presente trabalho estabelece os seguintes objetivos:

1.1.1 Objetivo Geral

Modernizar a base tecnológica do sistema exomaDB e expandir suas funcionalidades através da implementação de um módulo de visualização de dados genômicos, cuja arquitetura sirva como modelo de referência para garantir a futura manutenibilidade e escalabilidade de todo o sistema.

1.1.2 Objetivos Específicos

- Analisar criticamente o sistema exomaDB existente, identificando e documentando suas fragilidades arquiteturais e de gerenciamento de dependências.
- Implementar a modernização técnica do sistema, migrando o gerenciamento de dependências para o Maven e desenvolvendo um módulo de visualização de anotações genômicas com arquitetura em camadas (Repository, Service, Controller).
- Validar a nova arquitetura por meio de evidências práticas de melhoria em manutenibilidade, testabilidade e performance, consolidando-a como um padrão de referência para a evolução do sistema.

2 Fundamentação Teórica

2.1 Doenças Raras e Pesquisa Genômica

As doenças raras formam um vasto conjunto de condições médicas que, embora individualmente incomuns, afetam coletivamente uma parcela expressiva da população. No Brasil, a complexidade desses quadros é reconhecida oficialmente pela política de atenção do Ministério da Saúde (BRASIL. MINISTÉRIO DA SAÚDE, 2014). Estudos internacionais estimam que entre 3,5% e 5,9% da população mundial conviva com alguma doença rara, o que representa entre 263 e 446 milhões de pessoas (WAKAP et al., 2020).

Com mais de 6.000 tipos descritos, essas doenças são majoritariamente de origem genética (71,9%) e costumam se manifestar na infância (69,9%), englobando desde anomalias congênitas até erros inatos do metabolismo e deficiências intelectuais (WAKAP et al., 2020). O estudo dessas patologias foi revolucionado pelas tecnologias de sequenciamento de nova geração (NGS), que permitem analisar grandes volumes de dados genômicos de forma rápida e detalhada (GOODWIN; MCPHERSON; MCCOMBIE, 2016).

Como a maioria das mutações patogênicas conhecidas reside no exoma, a porção codificante do genoma, seu sequenciamento tornou-se uma estratégia de alto rendimento para o diagnóstico (BIESECKER; GREEN, 2014).

Contudo, os dados brutos gerados pelo NGS exigem ferramentas de bioinformática sofisticadas para serem processados, alinhados e interpretados (LESK, 2019). Bancos de dados como o ClinVar (LANDRUM et al., 2018) são vitais para classificar a patogenicidade das variantes encontradas, apoiando tanto o diagnóstico clínico quanto a pesquisa. Fica claro, portanto, que sistemas de informação robustos são indispensáveis para auxiliar os pesquisadores na complexa tarefa de interpretar dados genômicos.

2.2 Sistemas para Análise de Anotações Genômicas

As anotações genômicas agregam informações essenciais sobre genes e suas variantes, como localização cromossômica, função biológica e associação com doenças. Para a pesquisa em genética, especialmente no campo das doenças raras, o acesso rápido e a visualização clara desses dados são cruciais (WANG; LI; HAKONARSON, 2010).

Um sistema eficaz deve permitir que o pesquisador não apenas visualize, mas também filtre e ordene grandes volumes de informação de maneira intuitiva. Embora gráficos sejam úteis, uma apresentação em tabela bem estruturada costuma ser mais prática para a exploração sistemática de dados, oferecendo detalhes sem sobrecarregar

visualmente o usuário (MCLAREN et al., 2016). O principal desafio técnico, que orientou o desenvolvimento do módulo proposto no exomaDB, é justamente equilibrar a densidade de informações com a facilidade de uso.

2.3 Arquitetura de Software e Boas Práticas de Engenharia

O desenvolvimento de sistemas de bioinformática que sejam ao mesmo tempo robustos e fáceis de manter depende da aplicação de princípios de engenharia de software já consolidados.

2.3.1 Padrões de Design e Princípios SOLID

O padrão *Model-View-Controller* (MVC) é uma base para organizar aplicações interativas, separando as responsabilidades em três componentes: o Modelo (dados e regras de negócio), a Visão (interface do usuário) e o Controlador (que faz a mediação entre os outros dois) (KRASNER; POPE, 1988). Em sistemas como o exomaDB, que utiliza JavaServer Faces (JSF), esse padrão já faz parte da estrutura nativa do *framework*.

De forma complementar, os princípios SOLID (*Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion*), propostos por Martin (2003), funcionam como um guia para criar um software mais coeso, desacoplado e flexível:

- **Princípio da Responsabilidade Única (SRP):** Cada classe deve ter apenas um motivo para mudar.
- **Princípio Aberto-Fechado (OCP):** O software deve ser aberto para extensão, mas fechado para modificação.
- **Princípio da Substituição de Liskov (LSP):** Tipos derivados devem ser substituíveis por seus tipos base.
- **Princípio da Segregação de Interfaces (ISP):** Os clientes não devem ser forçados a depender de interfaces que não utilizam.
- **Princípio da Inversão de Dependência (DIP):** Módulos de alto nível não devem depender de módulos de baixo nível; ambos devem depender de abstrações.

A adesão a esses princípios é o que permite que um sistema de bioinformática evolua de forma sustentável, adaptando-se a novas descobertas sem comprometer sua estabilidade.

2.3.2 Anti-Padrões Arquiteturais (Anti-Patterns)

Em contraste com as boas práticas, existem os anti-padrões: soluções que parecem funcionar no curto prazo, mas que geram problemas sérios de manutenção e complexidade (BROWN et al., 1998). Identificar e corrigir esses anti-padrões é tão crucial quanto aplicar bons designs (FOWLER, 1999). Um dos mais comuns é a *God Class* (Classe Divina), uma classe que acumula responsabilidades demais, violando o SRP e tornando o código frágil e difícil de testar. Este foi, precisamente, o principal anti-padrão diagnosticado na arquitetura original do exomaDB.

2.3.3 Evolução Incremental e Débito Técnico

Modernizar um sistema legado, ainda mais um que está em uso, exige uma abordagem cuidadosa. A evolução incremental, defendida por autores como (FOWLER, 1999) e (SOMMERVILLE, 2011), propõe a introdução de melhorias graduais sem paralisar o sistema. Essa estratégia é ideal para gerenciar o débito técnico, termo cunhado por (CUNNINGHAM, 1992) para descrever o custo futuro de se optar por soluções fáceis no presente, permitindo que o software evolua de forma contínua e segura.

2.4 Tecnologias Empregadas

A nova arquitetura e as funcionalidades do exomaDB foram construídas sobre tecnologias consolidadas do ecossistema Java Enterprise:

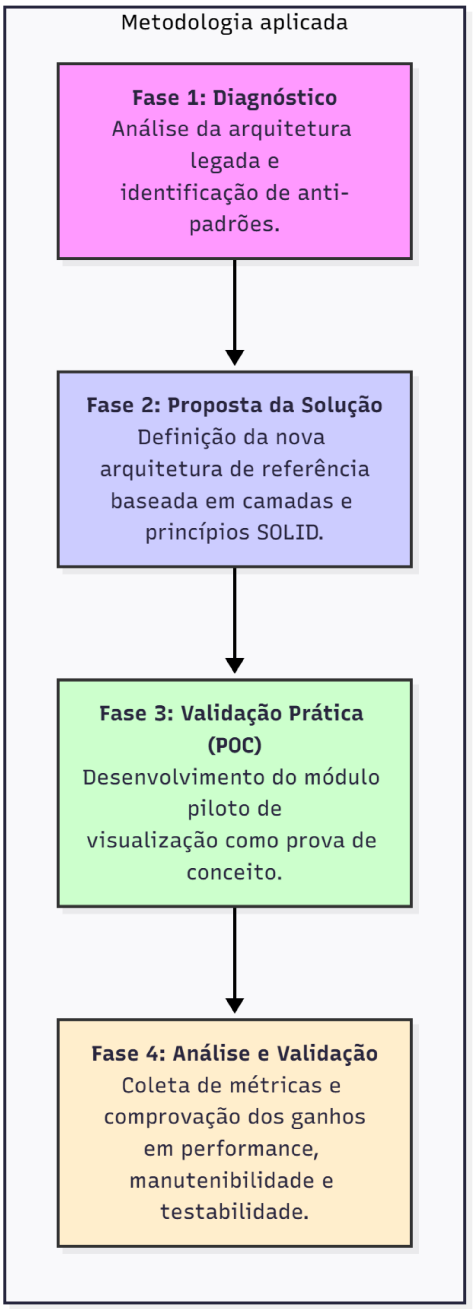
- **JavaServer Faces (JSF) e PrimeFaces:** O JSF é o *framework* padrão para construir interfaces web em Java baseadas em componentes (SAVAGE; KATZ, 2007). O PrimeFaces o estende com uma rica biblioteca de componentes avançados, perfeitos para criar as interfaces de visualização de dados complexos que o projeto exigia.
- **Java Persistence API (JPA) e Hibernate:** A JPA é a especificação padrão para mapeamento objeto-relacional (ORM), simplificando a interação com o banco de dados (KEITH; SCHINCARIOL, 2018). O Hibernate, sua implementação mais popular, oferece um *framework* robusto e confiável para a persistência de dados (BAUER; KING; GREGORY, 2015).

A escolha dessas tecnologias, guiada pelos princípios de arquitetura discutidos, permitiu construir um módulo que não apenas atende a uma necessidade imediata dos pesquisadores, mas também estabelece um modelo sólido para o futuro de todo o sistema exomaDB.

3 Metodologia

Este capítulo detalha a metodologia de engenharia de software utilizada para evoluir o sistema exomaDB. A abordagem foi estruturada em três fases principais: primeiro, um diagnóstico aprofundado da arquitetura original para identificar seus débitos técnicos (Seção 3.1); segundo, a definição de uma nova arquitetura de referência baseada em boas práticas de mercado para guiar a evolução do sistema (Seção 3.2); e terceiro, a validação prática dessa nova arquitetura através do desenvolvimento de um módulo piloto como Prova de Conceito (POC), focado na visualização de anotações genômicas (Seção 3.3). Por fim, são detalhados os critérios e testes usados para validar o sucesso da abordagem (Seção 3.4). A Figura 1 ilustra visualmente este fluxo metodológico.

Figura 1 – Fluxo Metodológico da Evolução do Sistema exomaDB



Fonte: Elaborado pelo autor (2025).

3.1 Diagnóstico da Arquitetura Original do exomaDB

O ponto de partida foi uma análise aprofundada da estrutura original do sistema. Esse diagnóstico foi crucial para mapear com clareza as lacunas funcionais, os débitos técnicos e as barreiras de arquitetura que dificultavam a manutenção e a criação de novas funcionalidades.

3.1.1 Análise Técnica Detalhada

A análise técnica foi conduzida via inspeção manual do código-fonte, versionado em um repositório Git. O projeto consiste em aproximadamente 15 classes e 8.000 linhas de código Java (versão 8). A análise focou em identificar pontos de alto acoplamento e baixa coesão, confirmando a necessidade de uma intervenção arquitetural.

A análise do código-fonte revelou desafios que impactavam diretamente a qualidade do software:

- **Complexidade Elevada:** A lógica de programação estava excessivamente centralizada. A classe `DatabaseOperations`, por exemplo, passava de 2.600 linhas, com métodos individuais que superavam 100 linhas e continham múltiplos níveis de aninhamento, tornando o código difícil de entender e modificar. Embora a complexidade do domínio seja alta, a concentração de toda a lógica de acesso a dados em uma única classe é um forte indicativo do anti-padrão *God Class*, que viola o Princípio da Responsabilidade Única e dificulta a manutenção e os testes.
- **Alto Acoplamento:** Havia um forte acoplamento entre as classes de domínio (POJOs) e a camada de acesso a dados. Foram encontradas mais de 70 instâncias diretas da classe `DatabaseOperations` (new DatabaseOperations()) espalhadas pelo código, criando uma dependência rígida que impedia a reutilização e, principalmente, a realização de testes.
- **Baixa Coesão:** Classes de modelo, como `participante`, misturavam responsabilidades distintas, agrupando atributos de domínio, lógica de interface de usuário (UI) e regras de negócio em um único arquivo.

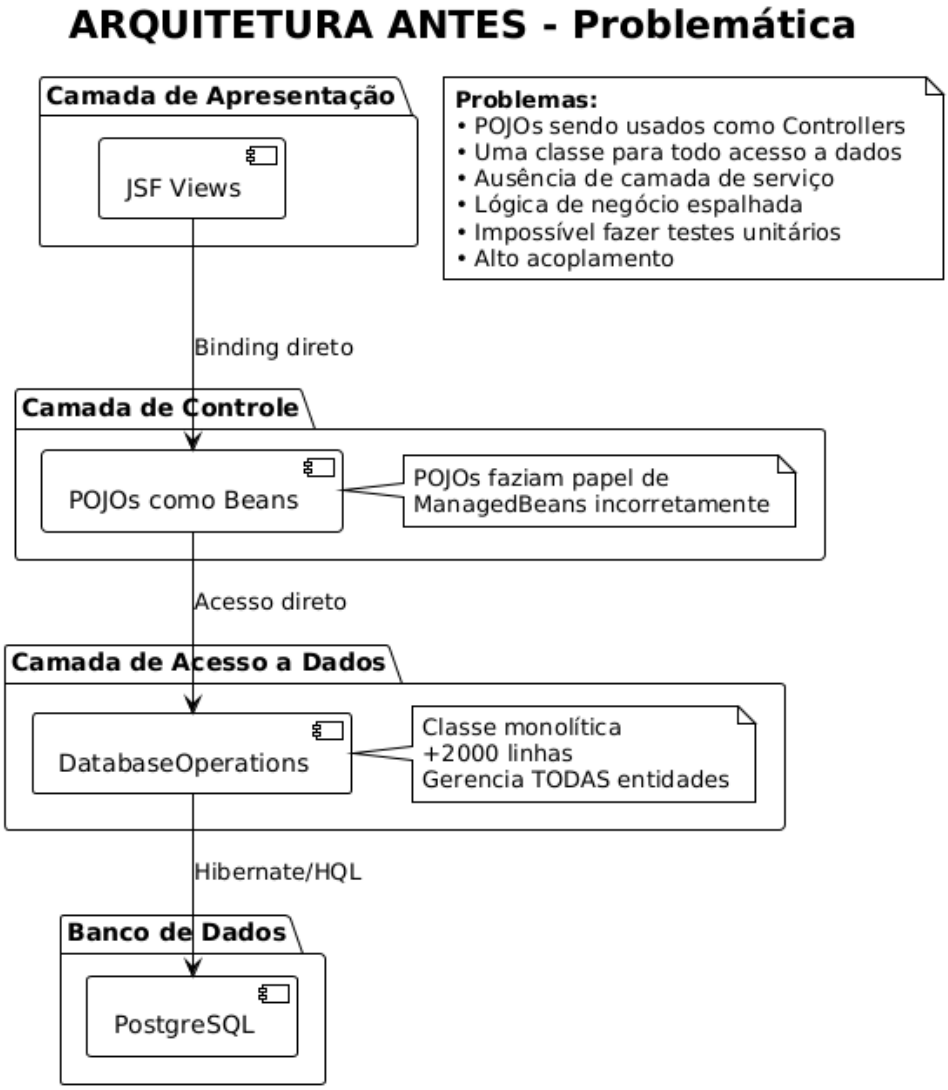
3.1.2 Anti-Padrões Arquiteturais Identificados

Essa análise confirmou a presença de vários anti-padrões que justificavam uma intervenção estrutural:

- **God Class:** A classe `DatabaseOperations` era um exemplo clássico, centralizando todo o acesso ao banco de dados para todas as entidades do sistema.
- **Feature Envy:** As classes de domínio frequentemente acessavam métodos de `DatabaseOperati`
  demonstrando "inveja"de funcionalidades que deveriam pertencer a uma camada de dados dedicada.
- **Shotgun Surgery:** Devido ao alto acoplamento, uma alteração simples, como adicionar um campo à classe `participante`, exigia modificações em múltiplos pontos do código, um processo lento e propenso a erros.

Na Figura 2 é ilustrada essa arquitetura original, destacando as conexões problemáticas e a centralização de responsabilidades que caracterizavam os anti-padrões encontrados.

Figura 2 – Arquitetura Anterior - Problemática



Fonte: Elaborado pelo autor (2025).

3.2 Definição da Arquitetura de Referência e Estratégia de Validação

Com base no diagnóstico, foi definida uma estratégia de evolução incremental para mitigar os riscos de uma reescrita completa do sistema, que está em produção. A estratégia consistiu em, primeiro, propor uma nova arquitetura de referência para o exomaDB, baseada em camadas (Visão, Controle, Serviço, Repositório) e nos princípios SOLID.

Para **validar a eficácia e os ganhos práticos** dessa nova arquitetura antes de aplicá-la ao restante do sistema legado, optou-se pelo desenvolvimento de um **módulo piloto**, funcionando como uma **Prova de Conceito (POC)**. O 'Módulo de Visualização de Anotações Genômicas' foi escolhido por ser uma funcionalidade de alta demanda pelos pesquisadores e suficientemente complexa para testar os limites da nova estrutura. O sucesso deste MVP (Produto Mínimo Viável) serviria como a evidência necessária para justificar a modernização futura dos demais módulos.

3.2.1 Ferramentas e Tecnologias

Para executar o projeto, foram selecionadas as seguintes ferramentas do ecossistema Java:

- **Linguagem:** Java
- **Interface de Usuário:** JavaServer Faces (JSF) com PrimeFaces
- **Persistência de Dados:** Java Persistence API (JPA) com Hibernate
- **Banco de Dados:** PostgreSQL
- **Gerenciamento de Dependências:** Maven
- **Testes:** JUnit 5 e Mockito
- **Ambiente de Desenvolvimento:** IntelliJ IDEA

3.2.2 Justificativa da Manutenção do Ecossistema Tecnológico

A decisão de evoluir o sistema mantendo o ecossistema tecnológico existente (Java, JSF/PrimeFaces) foi uma escolha de engenharia pragmática, alinhada ao contexto do trabalho. A abordagem de evolução incremental mitigou os riscos e os altos custos de uma reescrita completa, garantindo a continuidade da entrega de valor aos pesquisadores. Além disso, a arquitetura baseada em renderização no servidor (*Server-Side Rendering*) se mostrou ideal para o domínio da aplicação pois as operações de filtragem e processamento de grandes volumes de dados genômicos são computacionalmente intensivas. Manter essa carga no servidor otimiza o desempenho e garante que a interface do usuário permaneça leve e responsiva.

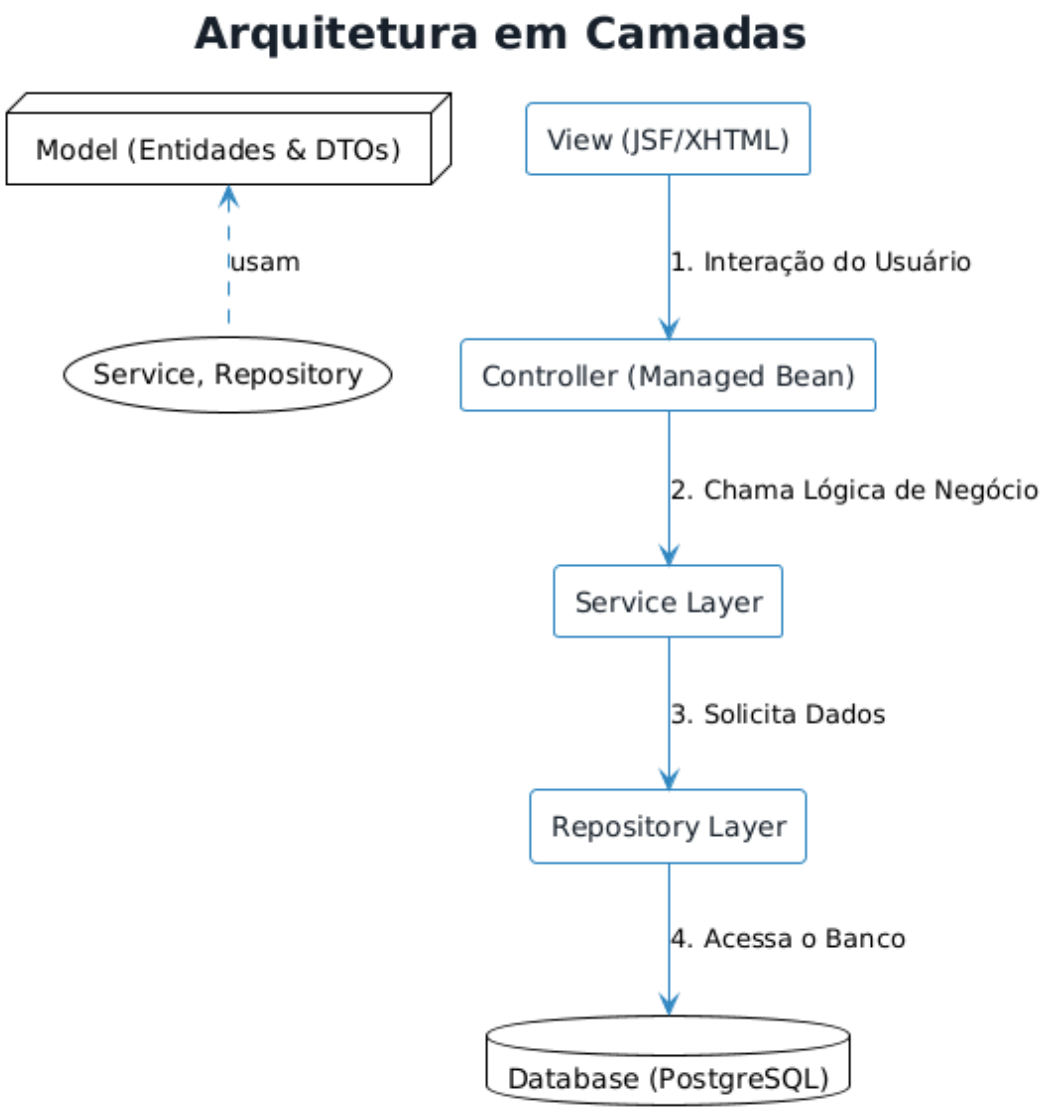
3.3 Desenvolvimento do Módulo Piloto como Prova de Conceito

Seguindo a estratégia de validação, o novo módulo de visualização de anotações genômicas foi desenvolvido. Este módulo serviu como a implementação prática da ar-

quitetura de referência, permitindo avaliar seus benefícios em um escopo controlado. A arquitetura foi estruturada em camadas bem definidas, conforme ilustrado na Figura 3.

3.3.1 Arquitetura em Camadas

Figura 3 – Arquitetura em Camadas



Fonte: Elaborado pelo autor (2025).

Na Figura 3 é ilustrada a nova arquitetura em camadas, que divide o sistema em cinco componentes lógicos com responsabilidades claramente distintas, facilitando a compreensão, a manutenção e a escalabilidade:

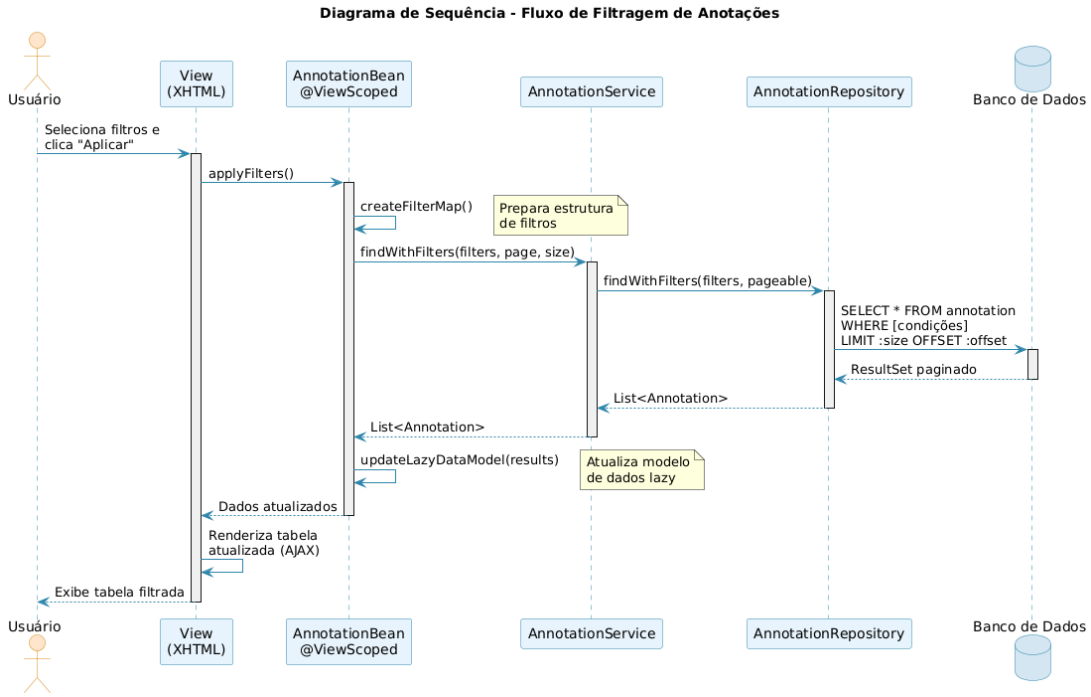
- **Visualização (View):** Responsável pela interface com o usuário, construída com páginas XHTML e componentes PrimeFaces, garantindo uma interação intuitiva e eficiente.

- **Controle (Controller):** Gerencia as interações do usuário, coordena as operações da aplicação e atua como intermediário entre a View e a camada de Serviço. É implementada através de Managed Beans do JSF, como o AnnotationBean.
- **Serviço (Service):** Encapsula a lógica de negócio e as regras específicas da aplicação (e.g., AnnotationService), orquestrando as operações e garantindo a aplicação das políticas de negócio.
- **Repositório (Repository):** Abstrai e implementa o acesso aos dados, utilizando JPA/Hibernate para interagir com o banco de dados (e.g., AnnotationRepository), isolando a lógica de persistência das demais camadas.
- **Modelo (Model):** Representa as entidades do domínio (e.g., Annotation, Consequence) através de classes POJO (*Plain Old Java Objects*), que encapsulam os dados e suas relações.

3.3.2 Implementação dos Padrões e Princípios SOLID

O padrão MVC e os princípios SOLID foram aplicados para garantir um código limpo e desacoplado. O fluxo de uma requisição, como a filtragem de anotações (Figura 4), ilustra bem essa separação: a Visão (XHTML) captura a ação do usuário e aciona o Controlador (AnnotationBean), que por sua vez delega a lógica para o Serviço (AnnotationService), que usa o Repositório (AnnotationRepository) para buscar os dados no banco.

Figura 4 – Diagrama de Sequência - Fluxo de Filtragem de Anotações



Fonte: Elaborado pelo autor (2025).

A aplicação dos princípios SOLID manifestou-se de forma concreta na nova arquitetura:

- **Princípio da Responsabilidade Única (SRP):** Cada camada agora possui uma única e bem definida responsabilidade. A classe AnnotationRepository é exclusivamente responsável pelo acesso aos dados, a AnnotationService encapsula apenas as regras de negócio, e a AnnotationBean gerencia unicamente o estado e as interações da interface de usuário. Isso eliminou a God Class (DatabaseOperations), que violava diretamente este princípio.
- **Princípio da Inversão de Dependência (DIP):** Este princípio é visível na camada de Controle. A classe AnnotationBean (módulo de alto nível) não depende de uma implementação concreta da camada de serviço (módulo de baixo nível), mas sim de uma abstração, injetada via @Inject. Isso desacopla as camadas e facilita a manutenção e os testes.

3.3.3 Otimização com Carregamento Lento (*Lazy Loading*)

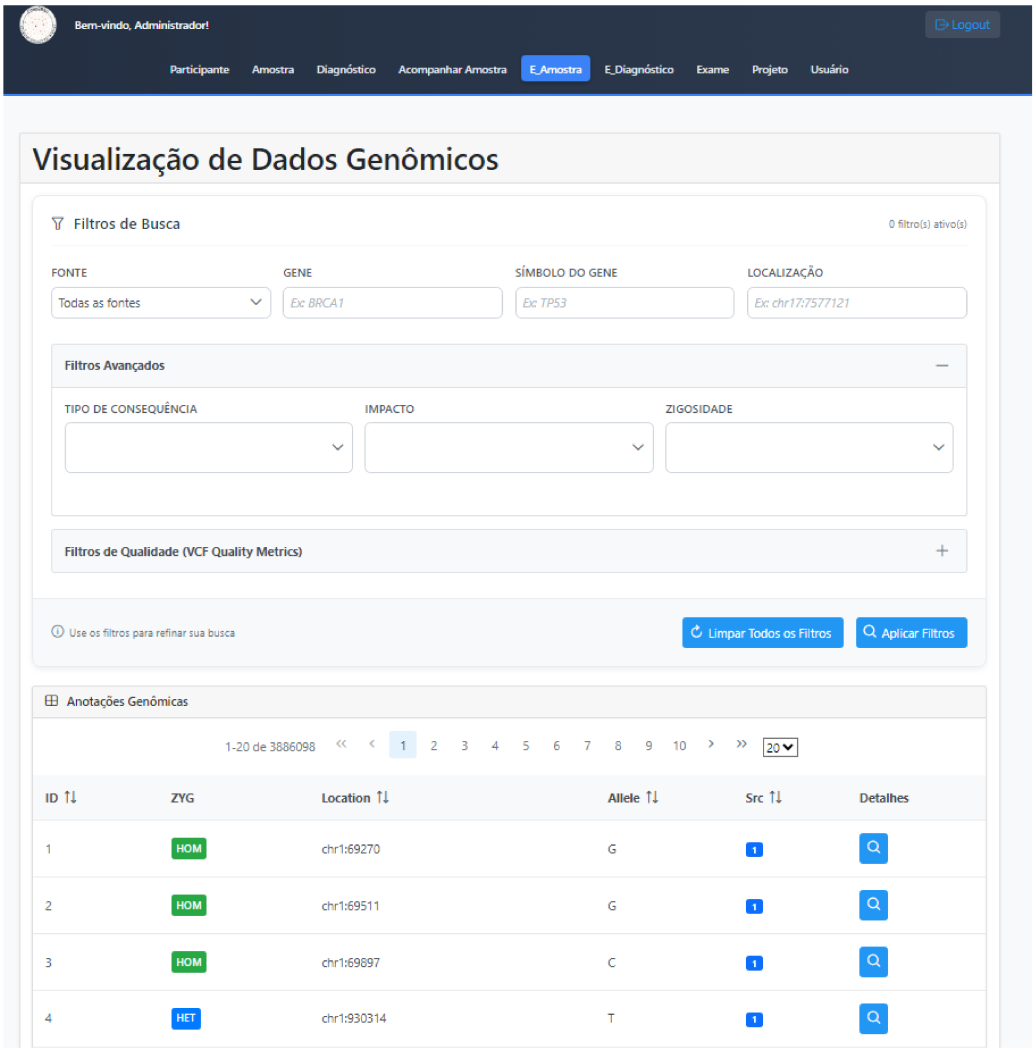
Para lidar com milhares de anotações genômicas sem comprometer a performance, foi implementado um modelo de carregamento sob demanda (*lazy loading*) com a classe `LazyDataModel` do PrimeFaces. Essa abordagem garante que apenas os dados da página

visível sejam carregados do banco de dados, otimizando o uso de memória e o tempo de resposta do sistema.

3.3.4 Interface de Usuário e Novas Funcionalidades

A interface do novo módulo foi projetada para otimizar o fluxo de trabalho dos pesquisadores, com uma tabela paginada e um painel de filtros expansível. Ao selecionar uma anotação, um painel lateral exibe detalhes de forma contextualizada, evitando que o usuário precise navegar para outras telas e tornando a análise mais fluida e eficiente (Figuras 5 e 6).

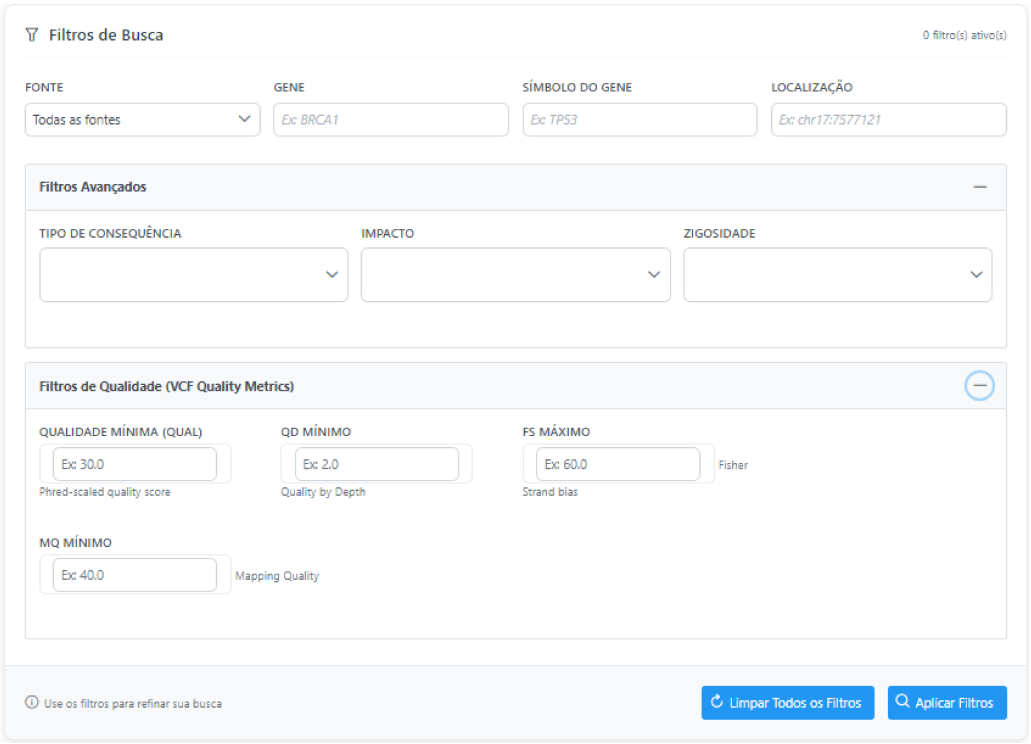
Figura 5 – Tela Principal do Módulo de Visualização de Anotações



Fonte: Elaborado pelo autor (2025).

A Figura 5 exibe a interface principal, destacando a tabela de anotações e os campos de filtro.

Figura 6 – Filtros Avançados



Fonte: Elaborado pelo autor (2025).

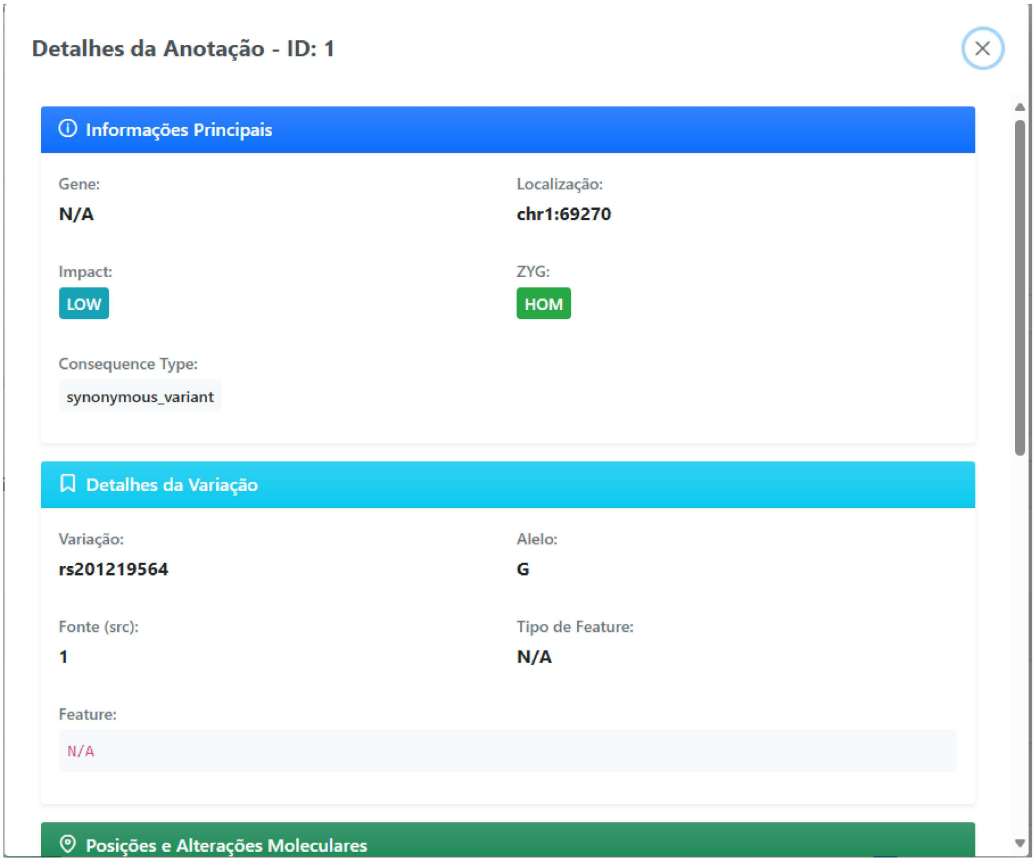
A Figura 6 detalha a seção de filtros de qualidade VCF (*Variant Call Format*) expandida, permitindo aos pesquisadores refinar ainda mais suas buscas.

As principais funcionalidades de filtragem implementadas incluem critérios como fonte de dados (src), gene, localização cromossômica, tipo de consequência, impacto e zigosidade (ZYG).

Esses critérios de filtragem não são arbitrários, pois refletem diretamente o fluxo de trabalho de um geneticista na busca por variantes patogênicas. Filtros como impacto (e.g., HIGH, MODERATE, LOW) permitem que o pesquisador priorize variantes com maior probabilidade de alterar a função de uma proteína. A zigosidade (homozigoto ou heterozigoto) é crucial para investigar padrões de herança de doenças. Finalmente, os filtros de qualidade VCF são indispensáveis para remover ruídos e artefatos técnicos do sequenciamento, garantindo maior confiabilidade nos resultados. A combinação desses filtros em uma interface única visa acelerar o processo de triagem de variantes, que é uma das etapas mais demoradas da análise genômica.

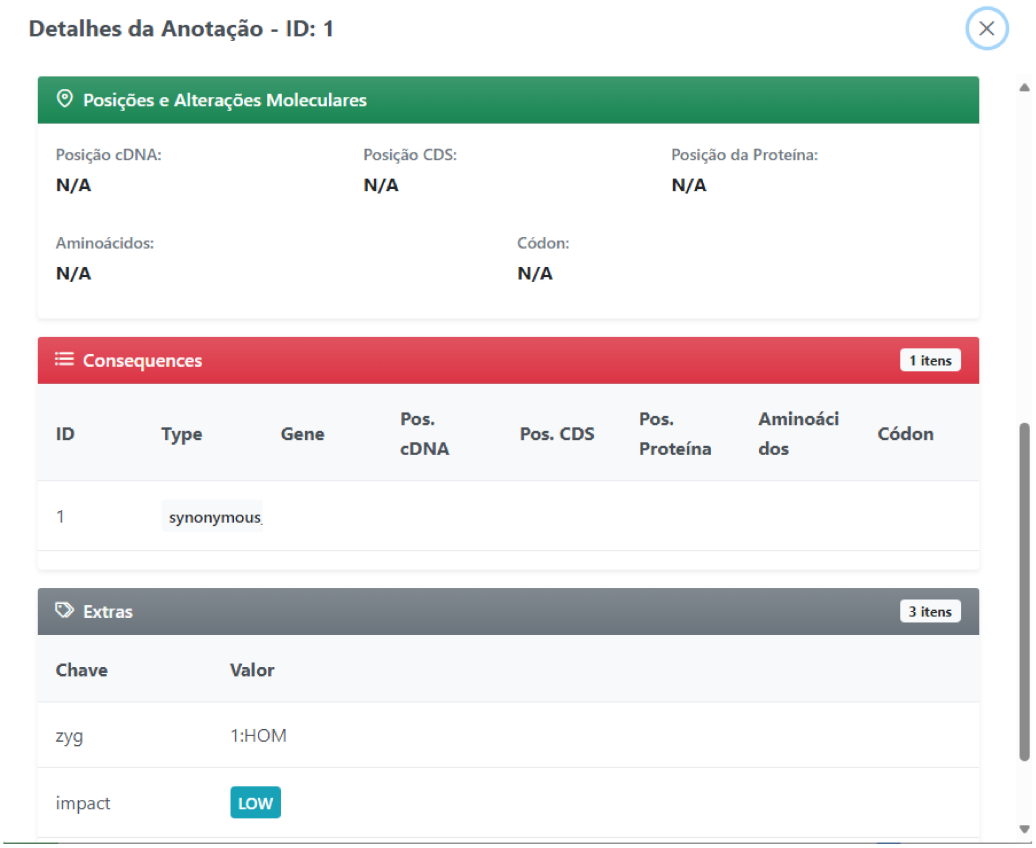
Ao selecionar uma anotação na tabela, um painel lateral é exibido, apresentando informações detalhadas de forma contextualizada. Esta abordagem elimina a necessidade de navegar para outras páginas, proporcionando uma experiência de usuário mais fluida e eficiente.

Figura 7 – Painel de Detalhes da Anotação - Seção Superior



Fonte: Elaborado pelo autor (2025).

Figura 8 – Painel de Detalhes da Anotação - Seção Inferior



Fonte: Elaborado pelo autor (2025).

As Figuras 7 e 8 ilustram o painel lateral com os detalhes da anotação selecionada, incluindo informações principais, detalhes da variação, posições moleculares, consequências e dados extras.

3.4 Critérios de Validação da Prova de Conceito

Para validar o sucesso do módulo piloto e, por consequência, da arquitetura de referência, foram estabelecidos três critérios objetivos de avaliação, cujos resultados são detalhados no Capítulo 4:

- **Viabilização da Testabilidade:** A nova arquitetura deveria permitir a criação de testes unitários automatizados. A métrica de sucesso foi a obtenção de uma alta cobertura de testes (acima de 80%) nas camadas de negócio.
- **Melhora na Manutenibilidade:** A adição de novas funcionalidades, como filtros de busca, deveria ser comprovadamente mais rápida. A métrica foi a redução percentual no tempo de desenvolvimento para essa tarefa recorrente.

- **Ganho de Performance:** A experiência do usuário deveria ser aprimorada no carregamento de grandes volumes de dados. A métrica foi a redução no tempo de resposta para carregar a tela principal de anotações.

Para avaliar a produtividade, foi cronometrado o tempo médio para implementar um novo filtro de busca na arquitetura legada e na nova. A performance foi medida utilizando as ferramentas de desenvolvedor do navegador, registrando o tempo de carregamento da tabela de dados em um cenário com mais de 3 milhões de anotações.

Para garantir a qualidade do código e validar a nova arquitetura, foi adotada uma estratégia de testes unitários estruturados com o padrão *Arrange-Act-Assert* (AAA). O foco foi testar o comportamento das camadas de Serviço e Repositório de forma isolada, usando *mocks* para simular suas dependências. As ferramentas utilizadas foram JUnit 5, Mockito e AssertJ.

A nova arquitetura, ao separar as responsabilidades, torna o código testável, uma capacidade que era praticamente inexistente na estrutura anterior.

4 Discussão de Resultados

A implementação do novo módulo e a modernização da base tecnológica do exomaDB trouxeram avanços práticos e estratégicos para o sistema. A validação do sucesso desta refatoração é demonstrada por meio de evidências concretas em quatro áreas críticas que impactam diretamente o ciclo de vida do software e a experiência do usuário.

4.1 Evidências de Melhoria Arquitetural

A transição da arquitetura monolítica para uma abordagem em camadas, guiada pelos princípios SOLID, foi a transformação mais impactante na qualidade interna do projeto.

4.1.1 Clareza Estrutural e Separação de Responsabilidades

A eliminação da "God Class" DatabaseOperations, que concentrava mais de 2.600 linhas de código, foi um dos principais resultados. A nova estrutura é visivelmente mais clara e coesa:

- **Camada de Repositório:** Cada entidade possui agora seu próprio Repository (ex: AnnotationRepository), focado exclusivamente no acesso ao banco de dados.
- **Camada de Serviço:** A lógica de negócio foi isolada em classes de Service (ex: AnnotationService).
- **Camada de Controle:** Os Controllers (Beans) foram simplificados, focando apenas na coordenação entre a interface e os serviços.

4.1.2 Produtividade e Facilidade de Manutenção

A implementação de um novo filtro de dados, uma tarefa recorrente, serve como um indicador prático da melhoria:

- **Arquitetura Legada:** Adicionar um filtro exigia modificações dispersas em múltiplas classes, um processo que consumia, em média, de 3 a 4 horas de desenvolvimento.
- **Nova Arquitetura:** As mudanças são localizadas e restritas a camadas específicas, permitindo que a mesma funcionalidade seja implementada em aproximadamente 30 minutos, uma redução de esforço superior a 85%.

4.2 Viabilização da Testabilidade

A arquitetura legada, com seu forte acoplamento, tornava a implementação de testes unitários impraticável. A nova estrutura, baseada em inversão de dependência, viabilizou a automação de testes, um pilar para a qualidade de software. Foi possível atingir uma cobertura de testes de aproximadamente 85% nas camadas críticas de Service e Repository, utilizando ferramentas como JUnit 5 e Mockito. Os 15% restantes correspondem majoritariamente a código de configuração e integração com o *framework*, cuja lógica é mais simples e com menor risco de falhas de negócio. Isso representa a introdução de uma prática de engenharia essencial que era impossível na estrutura anterior.

4.3 Impacto na Performance e Usabilidade para o Pesquisador

O impacto mais direto para o usuário final foi a melhoria drástica na performance do carregamento de dados. Operações que antes levavam aproximadamente 25 segundos, agora são concluídas em cerca de 5 segundos com a implementação do *lazy loading*, uma redução de 80% no tempo de espera do pesquisador.

4.4 Validação da Arquitetura como Modelo de Referência

A contribuição mais estratégica deste trabalho foi validar a nova arquitetura como um modelo viável para a evolução contínua do exomaDB. O sucesso do módulo de visualização, sua estabilidade, performance e testabilidade, serve como prova de conceito. A documentação e os padrões estabelecidos, detalhados no Apêndice A, funcionam agora como um guia seguro para modernizar os demais componentes legados do sistema, como os módulos de Participante e Amostra.

5 Conclusão

Este trabalho se propôs a evoluir o sistema exomaDB, uma ferramenta essencial para a pesquisa em doenças raras no LABGEN, atacando limitações técnicas que barravam seu crescimento. Através da modernização de sua base tecnológica e da implementação de um novo módulo de visualização de dados, o trabalho alcançou seus objetivos, entregando uma solução cuja melhoria foi validada por meio de evidências práticas e concretas. A nova arquitetura se provou consideravelmente mais robusta e manutenível, viabilizando pela primeira vez a prática de testes automatizados, que atingiram uma cobertura de 85% nas camadas de negócio. Aliado a isso, a performance para o usuário final foi notavelmente melhorada, consolidando a nova arquitetura como um modelo de referência para a evolução sustentável do sistema.

5.1 Retomada dos Objetivos e Contribuições

Os objetivos traçados no início do trabalho foram cumpridos de forma sistemática. A análise inicial diagnosticou as fragilidades da arquitetura legada, como a presença do anti-padrão *God Class*. Em seguida, a modernização técnica foi executada com sucesso, tanto na migração para o Maven quanto no desenvolvimento do novo módulo em camadas. Por fim, a nova arquitetura foi validada por meio de testes e documentada, consolidando-se como um modelo para o futuro do projeto.

As principais contribuições deste trabalho podem ser resumidas em:

1. **Contribuição Prática para o Projeto exomaDB:** Foi entregue aos pesquisadores do LABGEN um módulo funcional e de alta performance que otimiza a análise de variantes genéticas. A nova arquitetura resolveu gargalos de usabilidade, como o carregamento de dados em massa, através da implementação de *lazy loading*.

2. **Estudo de Caso de Evolução Incremental:** Este trabalho se consolida como um estudo de caso detalhado sobre a aplicação bem-sucedida de uma estratégia de evolução incremental em um sistema científico legado em produção. Demonstrou-se que é possível introduzir melhorias arquiteturais significativas de forma segura e com impacto positivo mensurável, sem a necessidade de uma reescrita completa.

3. **Validação de uma Arquitetura de Referência para o Sistema:** Mais do que criar uma arquitetura nova, este trabalho validou um modelo de referência para a evolução sustentável do ecossistema exomaDB. Ao aplicar princípios consolidados

de engenharia de software, foi criado um roteiro claro e testado, que servirá como guia para a modernização futura dos demais módulos, como Participante e Amostra.

5.2 Limitações e Trabalhos Futuros

Apesar dos resultados positivos, é fundamental reconhecer as limitações deste trabalho. A principal delas é o escopo limitado: a nova arquitetura foi aplicada a um único módulo, ainda que complexo. O grande desafio, a migração sustentável dos módulos legados Participante e Amostra, ainda permanece como um trabalho futuro. Este TCC estabeleceu a prova de conceito e o "caminho seguro", mas a jornada de modernização de todo o sistema está apenas no começo.

Adicionalmente, embora a cobertura de testes de 85% represente um avanço substancial, ela se concentrou nas camadas de serviço e repositório. A suíte de testes pode ser expandida para incluir testes de integração mais complexos e uma validação mais robusta da camada de visualização (View).

Estas limitações apontam para trabalhos futuros claros:

1. **Aplicação da Arquitetura de Referência:** Utilizar o modelo validado para refatorar de forma incremental os módulos legados de Participante, Amostra e Diagnóstico.

2. **Expansão da Cobertura de Testes:** Desenvolver uma suíte de testes de integração e de interface para garantir a qualidade de ponta a ponta dos módulos modernizados.

3. **Desenvolvimento de uma API REST:** Expor os dados do exomaDB por meio de uma API para permitir a integração com outras ferramentas de análise, desacoplando ainda mais o back-end do front-end.

4. **Integração com Bancos de Dados Externos:** Utilizando a API desenvolvida, conectar o sistema a bases de dados públicas como ClinVar e OMIM para enriquecer as anotações genômicas de forma automática.

5.3 Considerações Finais

Este trabalho reforça a importância de aplicar princípios sólidos de engenharia de software em domínios científicos. A evolução do exomaDB demonstrou que a qualidade da arquitetura de um sistema não é um mero detalhe técnico, mas um fator que habilita e acelera a pesquisa. Uma ferramenta robusta, intuitiva e manutenível permite que os cientistas dediquem seu tempo à sua verdadeira vocação: a descoberta científica.

A colaboração com os pesquisadores do LABGEN foi fundamental para o sucesso do trabalho e reafirma que as melhores soluções tecnológicas nascem do diálogo contínuo entre desenvolvedores e usuários. Em última análise, aprimorar ferramentas como o exomaDB é uma contribuição direta para o avanço no diagnóstico e na compreensão das doenças raras.

Referências

BAUER, C.; KING, G.; GREGORY, G. **Java Persistence with Hibernate**. [S.l.]: Manning Publications, 2015. Citado na página 12.

BIESECKER, L. G.; GREEN, R. C. Diagnostic clinical genome and exome sequencing. **New England journal of medicine**, Mass Medical Soc, v. 370, n. 25, p. 2418–2425, 2014. <https://doi.org/10.1056/NEJMra1312543>. Citado 2 vezes nas páginas 8 e 10.

BOYCOTT, K. M.; RATH, A.; CHONG, J. X.; HARTLEY, T.; ALKURAYA, F. S.; BAYNAM, G.; BROOKES, A. J.; BRUDNO, M.; CARRACEDO, A.; DUNNEN, J. T. den et al. International cooperation to enable the diagnosis of all rare genetic diseases. **The American Journal of Human Genetics**, Elsevier, v. 100, n. 5, p. 695–705, 2017. <https://doi.org/10.1016/j.ajhg.2017.04.003>. Citado na página 8.

BRASIL. MINISTÉRIO DA SAÚDE. **Doenças Raras: o que são, causas, tratamento, diagnóstico e prevenção**. 2014. Disponível em: http://portalms.saude.gov.br/. Acesso em: 10 dez. 2024. Citado 2 vezes nas páginas 8 e 10.

BROWN, W. J.; MALVEAU, R. C.; III, H. W. M.; MOWBRAY, T. J. **AntiPatterns: refactoring software, architectures, and projects in crisis**. [S.l.]: John Wiley & Sons, 1998. Citado na página 12.

CUNNINGHAM, W. The wycash portfolio management system. **ACM SIGPLAN OOPS Messenger**, ACM New York, NY, USA, v. 4, n. 2, p. 29–30, 1992. <https://doi.org/10.1145/157710.157715>. Citado na página 12.

DOHERTY, L. M.; BAMSHAD, M. J.; NICKERSON, D. A. Next-generation sequencing: an overview of current technologies and their clinical applications. **Genetics in medicine**, Nature Publishing Group, v. 22, n. 11, p. 1777–1786, 2020. Citado na página 8.

FOWLER, M. **Refactoring: Improving the Design of Existing Code**. [S.l.]: Addison-Wesley Professional, 1999. Citado 2 vezes nas páginas 9 e 12.

GOODWIN, S.; MCPHERSON, J. D.; MCCOMBIE, W. R. Coming of age: ten years of next-generation sequencing technologies. **Nature reviews genetics**, Nature Publishing Group, v. 17, n. 6, p. 333–351, 2016. <https://doi.org/10.1038/nrg.2016.49>. Citado na página 10.

HUANG, S.; WEIGEL, D.; BEACHY, R. N.; LI, J. Advances in genomics and genome editing for improving next generation plant breeding. **Current opinion in plant biology**, Elsevier, v. 54, p. 103–109, 2020. Citado na página 8.

KEITH, M.; SCHINCARIOL, M. **Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs**. [S.l.]: Apress, 2018. <https://doi.org/10.1007/978-1-4842-3420-4>. Citado na página 12.

KRASNER, G. E.; POPE, S. T. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. **Journal of object oriented programming**, v. 1, n. 3, p. 26–49, 1988. Citado na página 11.

LANDRUM, M. J.; LEE, J. M.; BENSON, M.; BROWN, G. R.; CHAO, C.; CHITIPIRALLA, S.; GU, B.; HART, J.; HOFFMAN, D.; JANG, W. et al. Clinvar: improving access to variant interpretations and supporting evidence. **Nucleic acids research**, Oxford University Press, v. 46, n. D1, p. D1062–D1067, 2018. <https://doi.org/10.1093/nar/gkx1153>. Citado na página 10.

LEHMAN, M. M. Programs, life cycles, and laws of software evolution. **Proceedings of the IEEE**, IEEE, v. 68, n. 9, p. 1060–1076, 1980. <https://doi.org/10.1109/PROC.1980.11805>. Citado na página 8.

LESK, A. M. **Introduction to bioinformatics**. [S.l.]: Oxford university press, 2019. <https://doi.org/10.1093/hesc/9780198794141.003.0001>. Citado na página 10.

MARTIN, R. C. **Agile Software Development: Principles, Patterns, and Practices**. [S.l.]: Prentice Hall PTR, 2003. Citado na página 11.

MCLAREN, W.; GIL, L.; HUNT, S. E.; RIAT, H. S.; RITCHIE, G. R.; THORMANN, A.; FLICEK, P.; CUNNINGHAM, F. The ensembl variant effect predictor. **Genome biology**, Springer, v. 17, n. 1, p. 1–14, 2016. <https://doi.org/10.1186/s13059-016-0974-4>. Citado na página 11.

SAVAGE, K. D.; KATZ, B. **JavaServer Faces in Action**. [S.l.]: Manning Publications Co., 2007. Citado na página 12.

SOMMERVILLE, I. **Software Engineering**. 9. ed. [S.l.]: Addison-Wesley, 2011. Citado 3 vezes nas páginas 8, 9 e 12.

WAKAP, S. N.; LAMBERT, D. M.; OLRY, A.; RODWELL, C.; GUEYDAN, C.; LANNEAU, V.; MURPHY, D.; CAM, Y. L.; RATH, A. Estimating cumulative point prevalence of rare diseases: analysis of the orphanet database. **European Journal of Human Genetics**, Nature Publishing Group, v. 28, n. 2, p. 165–173, 2020. <https://doi.org/10.1038/s41431-019-0508-0>. Citado 2 vezes nas páginas 8 e 10.

WANG, K.; LI, M.; HAKONARSON, H. Annovar: functional annotation of genetic variants from high-throughput sequencing data. **Nucleic acids research**, Oxford University Press, v. 38, n. 16, p. e164–e164, 2010. <https://doi.org/10.1093/nar/gkq603>. Citado na página 10.

WRIGHT, C. F.; FITZPATRICK, D. R.; FIRTH, H. V. Genetic diagnosis of developmental disorders in the ddd study: a scalable analysis of genome-wide research data. **The Lancet**, Elsevier, v. 392, n. 10154, p. 1305–1314, 2018. Citado na página 8.

Apêndices

APÊNDICE A – Guia do Desenvolvedor

Este guia contém a documentação técnica essencial para desenvolvedores que trabalharão no módulo de Visualização de Dados Genômicos.

A.1 Arquitetura e Padrões

O módulo segue uma arquitetura em camadas baseada nos princípios SOLID e no padrão MVC. A comunicação entre camadas deve seguir a hierarquia View → Controller → Service → Repository, e as dependências devem ser feitas por meio de interfaces, não de implementações concretas.

A.2 Estrutura de Código

A estrutura de pacotes reflete a separação de responsabilidades:

- `com.exoma.bean`: Managed Beans (Controllers)
- `com.exoma.service`: Interfaces e implementações da camada de negócio
- `com.exoma.repository`: Interfaces e implementações da camada de dados
- `com.exoma.pojo`: Entidades (Model)

A.3 Guias Práticos

A.3.1 Adicionando um Novo Filtro

Para adicionar um novo filtro (ex: por data), o processo envolve três passos:

1. No AnnotationBean (Controller): Adicionar o atributo do filtro.

```
// No AnnotationBean.java
private Date dateFilter;
// ... getters e setters
```

2. No AnnotationBean (Controller): Incluir o filtro no mapa de parâmetros.

```
// No m todo createFilterMap() do AnnotationBean
private Map<String, Object> createFilterMap() {
    Map<String, Object> filters = new HashMap<>();
    // ... outros filtros
```

```
    if (dateFilter != null) {
        filters.put("date", dateFilter);
    }
    return filters;
}
```

3. No AnnotationRepositoryImpl (Repository): Adicionar a lógica na consulta.

```
// No m todo findWithFilters() do AnnotationRepositoryImpl
if (filters.containsKey("date") && filters.get("date") != null) {
    queryBuilder.append(" AND DATE(a.creation_date) = :date");
    parameters.put("date", filters.get("date"));
}
```

A.3.2 Adicionando uma Nova Entidade

Crie a classe POJO com as anotações da JPA e o respectivo Repository.

1. Entidade AnnotationHistory.java:

```
@Entity
@Table(name = "annotation_history")
public class AnnotationHistory implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer historyId;

    // Outros campos, getters e setters
}
```

2. Interface AnnotationHistoryRepository.java:

```
public interface AnnotationHistoryRepository
    extends BaseRepository<AnnotationHistory, Integer> {

    List<AnnotationHistory> findByAnnotationId(Integer annotationId);
}
```

A.3.3 Exemplo de Teste Unitário com *Mocks*

Os testes unitários validam a lógica de negócio de forma isolada. Para isso, as dependências externas (como o repositório) são simuladas (*mockadas*).

1. Teste para AnnotationService:

```
// Teste para a camada de Servi o
@ExtendWith(MockitoExtension.class)
```

```
class AnnotationServiceTest {

    @Mock
    private AnnotationRepository annotationRepository; // Depend ncia
        simulada

    @InjectMocks
    private AnnotationServiceImpl annotationService; // Classe sob teste

    @Test
    void testFindById_shouldReturnAnnotation() {
        // Prepara o cen rio (Given)
        Annotation mockAnnotation = new Annotation();
        mockAnnotation.setId(1);
        when(annotationRepository.findById(1)).thenReturn(mockAnnotation);

        // Executa a a   o (When)
        Annotation result = annotationService.findById(1);

        // Verifica o resultado (Then)
        assertNotNull(result);
        assertEquals(1, result.getId());
    }
}
```

A.4 Boas Práticas

- **Otimização de Consultas:** Sempre utilize parâmetros nomeados para evitar injeção de SQL.

  Exemplo de parâmetros nomeados:

```
//     Correto - Uso de par metros nomeados
@Query("SELECT a FROM Annotation a WHERE a.gene = :geneName AND
    a.chromosome = :chr")
List<Annotation> findByGeneAndChromosome(@Param("geneName") String
    gene,
                                         @Param("chr") String
                                             chromosome);

//     Incorreto - Concatena  o direta (vulner vel a SQL
    injection)
String query = "SELECT * FROM annotation WHERE gene = '" + gene +
    "'";
```

- **Tratamento de Erros:** Capture exceções nas camadas inferiores e forneça mensagens claras na interface, sem expor *stack traces*.

Exemplo de tratamento de erros:

```
1  //     Correto - Tratamento adequado de exce  es
2  @Service
3  public class AnnotationServiceImpl implements AnnotationService {
4
5      public List<Annotation> findByFilter(Map<String, Object>
           filters) {
6          try {
7              return annotationRepository.findWithFilters(filters);
8          } catch (DataAccessException e) {
9              logger.error("Erro ao buscar anota  es: {}",
                   e.getMessage());
10             throw new ServiceException("Erro interno. Tente
                   novamente.");
11         }
12     }
13 }
14
15 //     Incorreto - Propagar exce  o t cnica para o usu rio
16 catch (SQLException e) {
17     throw e; // Exp e detalhes t cnicos desnecess rios
18 }
```