UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO INTERDISCIPLINAR EM CIÊNCIA E TECNOLOGIA

JOSÉ FERREIRA SOUSA NETO

DESENVOLVIMENTO DO BACK-END DE UM SISTEMA DE EMPRÉSTIMO DE MATERIAIS ACADÊMICOS NA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)

PAU DOS FERROS - RN
2025

JOSÉ FERREIRA SOUSA NETO

DESENVOLVIMENTO DO BACK-END DE UM SISTEMA DE EMPRÉSTIMO DE MATERIAIS ACADÊMICOS NA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel Interdisciplinar em Ciência e Tecnologia.

Orientador: Reudismam Rolim de Sousa, Prof. Dr.

PAU DOS FERROS - RN
2025

S725d Sousa Neto, José Ferreira.
Desenvolvimento do back-end de um sistema de empréstimos de materiais acadêmicos na universidade federal rural do semi-árido (UFERSA) / José Ferreira Sousa Neto. - 2025.
49 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência e Tecnologia, 2025.

1. Back-end. 2. Spring Boot. 3. Gestão universitária. 4. Empréstimo de materiais. 5. RESTful API. I. Sousa, Reudismam Rolim de, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

JOSÉ FERREIRA SOUSA NETO

DESENVOLVIMENTO DO BACK-END DE UM SISTEMA DE EMPRÉSTIMO DE MATERIAIS ACADÊMICOS NA UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO (UFERSA)

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência e Tecnologia.

Defendida em: **19/12/2025**.

BANCA EXAMINADORA

_____

Reudismam Rolim de Sousa, Prof. Dr. (UFERSA)
Presidente

_____

Glécia Mesquita Freire Fernandes, Me. (UFERSA)
Membro Examinador

_____

Lucas Francisco da Silva Souza, Bel. (UFERSA)
Membro Examinador

AGRADECIMENTOS

Gostaria de agradecer primeiramente à minha mãe, **Francisca de Sousa Dantas**, à minha vó **Maria de Sousa Dantas (falecida)**, ao meu tio **João Ferreira de Sousa** e à minha irmã **Karolayne Abrantes de Sousa**, que são as pessoas mais importantes na minha vida. Graças a eles, eu nunca parei de estudar e vou continuar estudando pelo futuro que me espera.

Quero agradecer ao meu amigo **Fabrício Andrey**, que esteve presente na minha vida desde que entrei nesse curso, por toda a felicidade, parceria e incentivo que já me proporcionou, e pelos projetos que realizamos juntos, e que ainda realizaremos.

Agradeço também ao meu amigo **Thomas Freitas**, que desde o início do curso tem sido um companheiro constante, trazendo alegria, apoio e colaboração em todos os projetos que desenvolvemos, e com quem pretendo continuar construindo muitas conquistas.

Também sou grato por todas as amizades que pude fazer no decorrer do curso, principalmente **Samara**, **Jennifer**, **Yslan**, **Mirna**, **Lucas**, **Allan**, **Berg** e **Victor** que me ajudaram a socializar e a moldar minha perspectiva de mundo.

Quero agradecer à banca examinadora, pela sua disposição de tempo e pela sua presença.

Agradeço também ao meu orientador, **Prof. Dr. Reudismam Rolim**, pela sua orientação, paciência e sabedoria, o que me ajudou muito no desenvolvimento deste trabalho.

RESUMO

A gestão eficiente de recursos materiais é crucial para o bom desempenho das atividades acadêmicas e administrativas na Universidade Federal Rural do Semi-Árido (UFERSA), Campus Pau dos Ferros. A ausência de um sistema centralizado e automatizado para o controle e empréstimo de equipamentos, como projetores, televisores e itens de laboratório, tem gerado dificuldades, como a subutilização de recursos e a falta de transparência. Este trabalho propõe o desenvolvimento do *back-end* de um sistema web para organizar e monitorar o empréstimo de materiais, implementando uma API RESTful que servirá como base para que, futuramente, alunos, professores e funcionários possam verificar a disponibilidade dos equipamentos, realizar reservas e gerenciar empréstimos de forma automatizada. Como metodologia para o desenvolvimento do sistema, foram aplicadas práticas consolidadas da Engenharia de Software, incluindo desenvolvimento iterativo com Spring Boot, implementação de segurança com autenticação JWT e testes automatizados. Para validar a robustez técnica da solução, foram conduzidos testes unitários e de integração, que comprovaram a confiabilidade e estabilidade dos *endpoints* desenvolvidos. Com a implementação deste *back-end*, a UFERSA poderá otimizar a utilização de seus materiais, aprimorar a gestão administrativa e criar um ambiente acadêmico mais eficiente e colaborativo, contribuindo para a modernização do gerenciamento de empréstimos no campus. O sistema também poderá ser expandido para outros campi da UFERSA e para outras instituições de ensino.

Palavras-chave: back-end; empréstimo de materiais; Spring Boot; gestão universitária.

ABSTRACT

The efficient management of material resources is crucial for the successful performance of academic and administrative activities at the Federal Rural University of Semi-Árido (UFERSA), Pau dos Ferros Campus. The absence of a centralized and automated system for controlling and lending equipment, such as projectors, televisions, and laboratory items, has created challenges such as underutilization of resources and lack of transparency. This work proposes the development of a web system back-end to organize and monitor material lending, implementing a RESTful API that will serve as the foundation for future development, allowing students, faculty, and staff to check equipment availability, make reservations, and manage loans in an automated manner. The system development methodology applied consolidated software engineering practices, including iterative development with Spring Boot, security implementation with JWT authentication, and automated testing. To validate the technical robustness of the solution, unit and integration tests were conducted, proving the reliability and stability of the developed endpoints. With the implementation of this back-end, UFERSA will be able to optimize the use of its materials, improve administrative management, and create a more efficient and collaborative academic environment, contributing to the modernization of loan management on campus. The system can also be expanded to other UFERSA campuses and other educational institutions.

Keywords: back-end; material loan; RESTful API; Spring Boot; university management.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CRUD	*Create, Read, Update, Delete*
DTO	*Data Transfer Object*
HTTP	*Hypertext Transfer Protocol*
JPA	*Java Persistence API*
JSON	*JavaScript Object Notation*
JWT	JSON *Web Token*
REST	*Representational State Transfer*

SUMÁRIO

1 INTRODUÇÃO

A operacionalização eficiente das atividades acadêmicas e administrativas em instituições de ensino superior depende criticamente do acesso a recursos materiais compartilhados. Na Universidade Federal Rural do Semi-Árido (UFERSA), Campus Pau dos Ferros, equipamentos como projetores, televisores, itens de laboratório e outros materiais especializados são essenciais para o desenvolvimento de atividades de ensino, pesquisa e extensão. No entanto, a natureza compartilhada desses recursos, disponibilizados principalmente por meio de empréstimos, impõe desafios significativos de gestão e controle (Melo *et al*., 2025).

A limitação na disponibilidade de certos itens, combinada com a ausência de um sistema integrado de gerenciamento, gera um cenário operacional complexo. Materiais frequentemente tornam-se indisponíveis nos momentos de maior demanda, comprometendo o andamento de atividades acadêmicas essenciais e gerando frustração entre discentes, docentes e técnicos administrativos. Adicionalmente, a falta de mecanismos automatizados para controle pode resultar na subutilização de alguns recursos enquanto outros são excessivamente demandados, caracterizando uma alocação ineficiente do patrimônio institucional (Melo *et al*., 2025).

O modelo atual, baseado em processos manuais e descentralizados, apresenta vulnerabilidades significativas: dificulta a transparência, aumenta a suscetibilidade a erros humanos e consome tempo considerável dos servidores envolvidos na organização (Melo *et al*., 2025). Esta realidade evidencia a necessidade urgente de uma solução tecnológica robusta que modernize e otimize a gestão de empréstimos na instituição.

Diante deste contexto, o presente trabalho propõe o desenvolvimento do *back-end* de um sistema web especializado para gestão de empréstimos de materiais acadêmicos. Diferente de abordagens que priorizam inicialmente a interface do usuário, este projeto concentra-se na construção das fundações tecnológicas que garantirão a segurança, confiabilidade e escalabilidade do sistema como um todo.

A solução técnica consiste em uma API RESTful abrangente desenvolvida em Java Spring Boot, que encapsula toda a lógica de negócio necessária para operacionalizar o fluxo completo de empréstimos. O sistema implementa um mecanismo robusto de autenticação e autorização baseado em JWT, com validação específica para e-mails institucionais da UFERSA, garantindo acesso restrito à comunidade acadêmica.

Por meio de *endpoints* especializados, a API gerencia todo o ciclo de vida dos empréstimos, desde o cadastro de usuários e materiais até o controle de reservas, devoluções e geração de relatórios. A arquitetura adota boas práticas consolidadas da Engenharia de Software, incluindo separação em camadas, uso de DTOs para transferência de dados, mapeamento entre entidades, validações centralizadas e tratamento padronizado de exceções.

O objetivo deste *back-end* transcende a mera automação de processos, posicionando-se como um facilitador para a transformação digital na gestão patrimonial da UFERSA. Ao estabelecer uma base tecnológica sólida e bem documentada via OpenAPI/Swagger, este projeto não apenas resolve problemas operacionais imediatos, mas também cria as condições para implementação de políticas de compartilhamento responsável de recursos, otimização do uso do patrimônio institucional e fortalecimento da transparência administrativa.

A implementação bem-sucedida desta API representa um avanço significativo na modernização da infraestrutura tecnológica do Campus Pau dos Ferros, alinhando a UFERSA às melhores práticas de gestão universitária e criando as bases para um ambiente acadêmico mais eficiente, colaborativo e inovador. O *back-end* proposto foi concebido para permitir futura expansão para outros campi da UFERSA e adaptação para outras instituições de ensino, promovendo o compartilhamento de soluções tecnológicas entre comunidades acadêmicas.

1.1 Justificativa

O desenvolvimento do *back-end* para um sistema de gestão de empréstimos de materiais na UFERSA é justificado por razões técnicas, operacionais e estratégicas que evidenciam a relevância e urgência deste projeto para a instituição. A complexidade inerente à administração de recursos materiais em uma universidade com ampla diversidade de cursos e atividades acadêmicas demanda uma solução tecnológica com fundamentação arquitetural sólida.

Primariamente, a gestão de materiais na UFERSA enfrenta desafios críticos relacionados à escalabilidade e confiabilidade dos processos. O modelo atual, baseado em procedimentos manuais e sistemas descentralizados, demonstra-se insuficiente para atender à demanda crescente por equipamentos como projetores, televisores e recursos de laboratório (Melo *et al*., 2025). Esta limitação técnica resulta em falta de transparência, alocação inadequada de recursos e sobrecarga operacional para os servidores envolvidos.

A opção pelo desenvolvimento de uma API RESTful especializada, em contraposição a uma solução *front-end* inicial, fundamenta-se na necessidade de estabelecer bases tecnológicas robustas antes da implementação de interfaces de usuário. Enquanto soluções superficiais focadas apenas na apresentação podem oferecer melhorias imediatas na experiência do usuário, a robustez do *back-end* — responsável pela lógica de negócio, segurança dos dados e integração com sistemas — é determinante para a longevidade e confiabilidade de qualquer sistema de gestão.

Do ponto de vista da Engenharia de Software, este projeto justifica-se pela implementação de uma arquitetura em camadas, facilitando a manutenção e evolução do sistema. A adoção de práticas consolidadas como uso de DTOs para transferência de dados, mapeamento entre entidades, validações centralizadas e tratamento padronizado de exceções assegura a qualidade do código e reduz a taxa de erros em produção.

Tecnicamente, a implementação de um sistema de autenticação e autorização baseado em JWT com validação específica para e-mails institucionais da UFERSA (@alunos.ufersa.edu.br e @ufersa.edu.br) garante a segurança do patrimônio acadêmico, permitindo o acesso restrito à comunidade universitária. Esta abordagem previne o uso indevido de recursos e assegura a rastreabilidade das operações realizadas.

Sob a perspectiva da eficiência operacional, a automação proporcionada por um *back-end* dedicado é fundamental para resolver os entraves dos processos manuais. A implementação de regras de negócio automatizadas para controle de prazos, verificação de disponibilidade em tempo real e gestão de conflitos de reserva não apenas reduz a carga de trabalho dos técnicos administrativos, como também minimiza erros humanos, aumentando a precisão e confiabilidade das informações.

A decisão pela *stack* tecnológica Java Spring Boot alinha-se com as melhores práticas de desenvolvimento empresarial, oferecendo maturidade, estabilidade e amplo suporte comunitário. A integração com PostgreSQL assegura a persistência confiável dos dados, enquanto a documentação via OpenAPI/Swagger facilita a futura integração com front-ends e outros sistemas institucionais.

Estrategicamente, a existência de um *back-end* funcional e bem documentado acelera significativamente o desenvolvimento de interfaces futuras, estabelecendo um contrato claro (a API) que pode ser utilizado por diferentes equipes. Esta abordagem incremental permite a validação técnica da solução antes de investimentos em componentes de apresentação, mitigando riscos de projeto.

Além dos benefícios técnicos imediatos, este projeto promove uma cultura de inovação tecnológica na instituição, alinhando a UFERSA às tendências de transformação digital no ensino superior. A arquitetura proposta serve como referência para futuros desenvolvimentos, estabelecendo padrões de qualidade e boas práticas que podem ser replicados em outros projetos institucionais.

Por fim, do ponto de vista acadêmico, este trabalho representa a aplicação prática de conceitos fundamentais da Ciência da Computação e Engenharia de Software em um contexto real, demonstrando a capacidade de traduzir necessidades institucionais em soluções tecnológicas eficazes.

Dessa forma, a justificativa para o desenvolvimento deste *back-end* é amplamente embasada na necessidade de uma infraestrutura tecnológica sólida, na modernização dos processos administrativos e na promoção de uma cultura de excelência técnica e inovação na UFERSA.

1.2 Objetivos

Os objetivos gerais e específicos do trabalho podem ser vistos a seguir.

1.2.1 Objetivo geral

O objetivo principal deste trabalho é desenvolver o *back-end* de um sistema Web para gerenciamento de empréstimos de materiais acadêmicos na UFERSA, por meio da implementação de uma API RESTful robusta e segura, que sirva como base tecnológica para futuras interfaces e otimize a gestão de recursos institucionais.

1.2.2 Objetivos específicos

Para atingir o objetivo principal, definimos estes objetivos específicos:

- **Levantar requisitos do sistema**: Realizar um levantamento detalhado dos requisitos funcionais e não funcionais junto aos potenciais usuários (estudantes, professores e

funcionários administrativos) e aos gestores dos setores envolvidos, identificando as regras de negócio e processos que a API deve contemplar.

- **Projetar a arquitetura da solução**: Definir um *back-end*, incluindo a modelagem do banco de dados, a diagramação dos *endpoints* da API e a especificação das tecnologias a serem utilizadas (Java Spring Boot, PostgreSQL e Spring Security).
- **Implementar as entidades e relacionamentos básicos**: Programar e implementar as entidades principais do sistema (Usuário, Material, Empréstimo) e seus relacionamentos no banco de dados, garantindo a integridade das informações e a consistência dos dados.
- **Desenvolver os *endpoints* da API**: Criar os *endpoints* da API RESTful para operações essenciais (CRUD - Criar, Ler, Atualizar e Deletar) e para as regras de negócio específicas, tais como solicitação de empréstimo, aprovação, devolução e consulta de disponibilidade.
- **Implementar sistema de autenticação e autorização**: Integrar um mecanismo seguro de autenticação baseado em JWT (JSON *Web Tokens*) para controle de acesso, definindo níveis de permissão para os diferentes perfis de usuário (estudante, professor, administrador) e validando e-mails institucionais.
- **Realizar testes técnicos**: Conduzir testes unitários e de integração para validar o correto funcionamento de cada componente do *back-end* e da API, assegurando a confiabilidade, estabilidade e segurança do sistema desenvolvido.
- **Documentar a API**: Elaborar documentação clara e compreensível da API utilizando OpenAPI/Swagger, detalhando cada *endpoint*, seus parâmetros, possíveis respostas e exemplos de uso, para facilitar o futuro desenvolvimento do front-end e a integração com outros sistemas.

1.3 Organização do trabalho

O presente trabalho está estruturado da seguinte maneira: no Capítulo 2, é apresentado o embasamento teórico e os trabalhos relacionados a esta pesquisa; no Capítulo 3, é detalhada a proposta da solução, incluindo o levantamento de requisitos, a modelagem arquitetural e a estrutura de implementação; no Capítulo 4, são apresentadas os resultados de avaliação da arquitetura, com base em cenários de uso e na implementação dos artefatos arquiteturais; por fim, o Capítulo 5 reúne as considerações finais do projeto, bem como sugestões de melhorias e direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos teóricos e tecnológicos que embasam o desenvolvimento do *back-end* para o sistema de gestão de empréstimos de materiais acadêmicos na UFERSA. Os tópicos discutidos incluem arquitetura de software, desenvolvimento de APIs RESTful, segurança em sistemas web, persistência de dados e metodologias ágeis de desenvolvimento.

2.1 Gestão de recursos em instituições de ensino

A gestão eficiente de recursos materiais constitui um pilar fundamental para o funcionamento adequado de instituições educacionais. Segundo Chiavenato (2010), a administração de recursos envolve planejamento, organização, direção e controle, permitindo o uso racional e eficaz dos bens disponíveis. Em ambientes universitários, em que a demanda por materiais é intensa e diversificada, a ausência de controles adequados pode resultar em subutilização, desperdícios e conflitos na alocação de recursos (Alves; Ribeiro, 2016).

No contexto específico da UFERSA, a carência de um sistema informatizado centralizado torna os processos de empréstimo e controle de materiais vulneráveis a erros humanos, retrabalho e insatisfação dos usuários. A implementação de soluções tecnológicas robustas para essa gestão torna-se, portanto, estratégia imperativa para garantir transparência, rastreabilidade e otimização dos recursos disponíveis (Melo *et al*., 2025).

2.2 Java

A linguagem Java, desenvolvida pela Sun Microsystems (atualmente Oracle), constitui uma das plataformas de desenvolvimento mais consolidadas e amplamente adotadas no cenário empresarial. Caracterizada pelo princípio "*write once, run anywhere*" (escreva uma vez, execute em qualquer lugar), Java permite que aplicações compiladas executem em qualquer plataforma que possua uma Java Virtual Machine (JVM) instalada (Oracle, 2023).

Segundo a especificação da linguagem Java SE 21, a plataforma oferece recursos avançados como orientação a objetos, gerenciamento automático de memória (*garbage collection*), tratamento de exceções, tipos genéricos e programação concorrente, que facilitam o desenvolvimento de aplicações robustas e escaláveis (Oracle, 2023). A maturidade do ecossistema Java, com vasta biblioteca padrão e comunidade ativa, torna-a particularmente adequada para o desenvolvimento de sistemas corporativos como o proposto neste trabalho.

No contexto deste projeto, a escolha pela versão Java 21 justifica-se pelos recursos de modernização da linguagem, melhorias de *performance* e suporte estendido, alinhando-se com as melhores práticas contemporâneas de desenvolvimento de software (Oracle, 2023).

2.3 *Framework Spring*

O desenvolvimento *back-end* representa a camada do software que executa no servidor, sendo responsável pela lógica de negócio, processamento de dados, autenticação e comunicação com o banco de dados. Diferente do *front-end*, que lida com a apresentação, o *back-end* constitui o cerne funcional da aplicação (Pressman; Maxim, 2020).

O uso de *frameworks* modernos como Spring Boot para Java acelera o desenvolvimento ao fornecer estruturas pré-estabelecidas para configuração, segurança e integração com bancos de dados, seguindo boas práticas e padrões de projeto (Gamma *et al*., 2000). A escolha do Spring Boot impacta diretamente na produtividade, *performance* e manutenibilidade do sistema, oferece configuração automática, *embedded server* e ecossistema rico de dependências gerenciadas (Spring.io, 2024).

2.4 Arquitetura de software e padrão REST

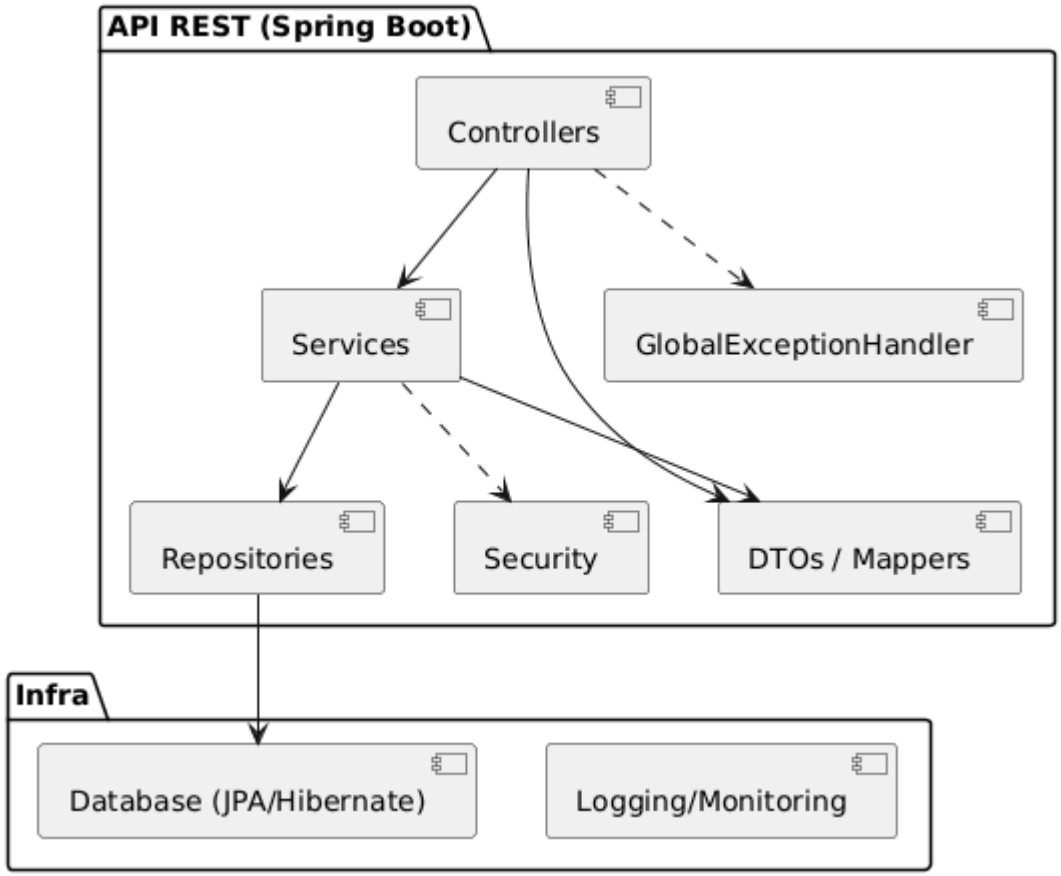
A arquitetura de software define a organização fundamental de um sistema, encapsulando suas partes constituintes e as relações entre elas. Para sistemas distribuídos como o proposto, o padrão arquitetural REST (*Representational State Transfer*), definido por Fielding (2000), tornou-se padrão de facto para desenvolvimento de APIs web. REST descreve um conjunto de *constraints* arquiteturais que, quando seguidas, permitem a criação de sistemas escaláveis, de desempenho otimizado e manutenção simplificada.

Conforme Fowler (2002), a adoção do padrão REST é crucial para o desenvolvimento de APIs previsíveis e bem estruturadas, que facilitam a integração com *front-ends* e outros serviços. No contexto deste projeto, a arquitetura RESTful permite a criação de *endpoints* claramente definidos para operações sobre recursos como usuários, materiais e empréstimos, seguindo os verbos HTTP padronizados (GET, POST, PUT, DELETE).

Cada verbo HTTP corresponde a uma operação específica e semântica sobre os recursos: o método GET é utilizado para recuperar dados de um recurso específico ou de uma coleção, sem causar efeitos colaterais no servidor; POST é empregado para criar um novo recurso, geralmente enviando dados no corpo da requisição; PUT serve para atualizar por completo um recurso existente, substituindo seus dados; e DELETE remove um recurso

identificado. Essa clareza nas operações garante que a API seja intuitiva e consistente, promovendo uma comunicação eficiente entre cliente e servidor.

Figura 1 - Arquitetura da API REST do Back-end



Fonte: Autoria própria (2025)

2.5 Persistência de dados com JPA e PostgreSQL

A modelagem de dados é uma etapa crítica que define como as informações do sistema serão estruturadas e relacionadas. Um projeto bem elaborado, seguindo as formas normais para evitar redundâncias e inconsistências, é fundamental para a integridade e *performance* do sistema (PostgreSQL, 2024).

O Spring Data JPA simplifica o acesso a dados por meio do padrão Repository, abstraindo a complexidade das operações SQL e permitindo foco na lógica de negócio. A definição clara de entidades, atributos e relacionamentos, representada por um Diagrama Entidade-Relacionamento, serve como planta para a implementação física no Sistema Gerenciador de Banco de Dados escolhido (PostgreSQL, 2024).

2.6 Metodologias de desenvolvimento ágil

Para gerenciar a complexidade do desenvolvimento de software, metodologias ágeis como Scrum são amplamente adotadas. Elas promovem processo iterativo e incremental, com ciclos de desenvolvimento curtos (*sprints*), que permitem adaptações rápidas baseadas em feedback contínuo (Beck *et al*., 2001).

A aplicação de práticas ágeis neste projeto facilita o acompanhamento do progresso, a gestão de tarefas e a garantia de que o *back-end* desenvolvido atenda efetivamente aos requisitos levantados. A abordagem iterativa permite refinamentos progressivos na arquitetura e implementação dos componentes do sistema.

2.7 Qualidade de código e boas práticas

A manutenibilidade do software é diretamente influenciada pela qualidade do código produzido. Conforme Martin (2009), princípios de código limpo incluem significados expressivos, funções pequenas e focadas, formatação consistente e tratamento adequado de erros.

A implementação de testes automatizados com JUnit assegura a qualidade e confiabilidade do *back-end*, permitindo validação contínua do funcionamento dos componentes e prevenindo regressões (JUnit, 2024). Testes unitários verificam o correto funcionamento de unidades individuais de código, enquanto testes de integração validam a comunicação entre diferentes módulos do sistema.

2.8 Tratamento de erros e exceções

O tratamento de erros da API é centralizado em um manipulador global que converte exceções em respostas HTTP padronizadas. Isso garante a consistência das respostas e evita a exposição de informações sensíveis.

Erros são mapeados para códigos HTTP semânticos:

- Validação de entrada → Código 400 (*Bad Request*);
- Recurso não encontrado → Código 404 (*Not Found*);
- Conflitos de estado/integridade → Código 409 (*Conflict*);
- Falhas de autenticação → Código 401/403;
- Erros inesperados → Código 500 (*Internal Server Error*).

O corpo das respostas de erro usa um formato único (*ErrorResponse*), contendo pelo menos: *status*, *error*, *message*, *path* e *timestamp*. Em validações, message agrega as mensagens por campo (ex.: "cpf: CPF deve ter 11 dígitos").

Regras de negócio são validadas na camada de serviço e lançam exceções controladas (ou *ResponseStatusException*); quando necessário, o manipulador as transforma em *ErrorResponse* com o código apropriado. Operações críticas usam transação (*@Transactional*) para garantir *rollback* em caso de falha.

2.9 Ferramentas de teste e documentação de APIs

O desenvolvimento de APIs RESTful demanda ferramentas especializadas para teste e documentação durante o ciclo de desenvolvimento. Entre as soluções disponíveis, o Postman destaca-se como plataforma colaborativa completa para desenvolvimento de APIs, permitindo a criação, teste, documentação e monitoramento de *endpoints* RESTful (Postman, 2024).

Segundo a documentação oficial do Postman (2024), a ferramenta oferece funcionalidades essenciais como:

- Coleções de requisições organizadas por funcionalidade.
- Ambiente de testes com variáveis configuráveis.
- Automação de testes de integração.
- Geração de documentação interativa.
- Simulação de diferentes cenários de autenticação.

No contexto deste projeto, o Postman foi utilizado para validar todos os *endpoints* implementados, assegurando o correto funcionamento dos mecanismos de autenticação JWT, validações de negócio e fluxos completos de empréstimo e devolução de materiais. A ferramenta também facilitou a documentação preliminar da API durante o desenvolvimento.

Complementarmente, a documentação OpenAPI/Swagger integrada ao Spring Boot fornece interface interativa para exploração dos *endpoints*, seguindo o padrão *industry* para documentação de APIs RESTful (OpenAPI Initiative, 2023).

3 METODOLOGIA

Para o desenvolvimento do *back-end* do sistema de empréstimos de materiais acadêmicos proposto neste trabalho, foi adotada uma abordagem baseada em práticas consolidadas da Engenharia de Software, com foco no desenvolvimento iterativo e incremental. A metodologia aplicada integra elementos de processos ágeis com técnicas específicas de desenvolvimento de APIs RESTful, seguindo um ciclo estruturado em três fases principais: planejamento, implementação e validação.

A abordagem metodológica foi orientada pelos princípios do desenvolvimento ágil (Beck *et al*., 2001), que privilegiam a entrega contínua de valor, adaptação a mudanças e *feedback* constante. Diferente de metodologias tradicionais em cascata, esta abordagem permitiu refinamentos progressivos na arquitetura e implementação, alinhando-se com a natureza evolutiva do desenvolvimento de software moderno.

3.1 Arquitetura do sistema

A arquitetura da aplicação segue o padrão em camadas (*layered architecture*), separando responsabilidades em *Presentation* (*Controllers*), *Business* (*Services*), *Persistence* (*Repositories*) e *Data Transfer* (DTOs / *Mappers*). Essa organização facilita testabilidade, manutenção e evolução do sistema, além de permitir a aplicação de políticas transversais (validação, segurança, *logging*) de forma centralizada.

Principais componentes e responsabilidades:

- ***Controller***: expõe os *endpoints* REST e trata requisições/respostas. Validações básicas de entrada são aplicadas via *Bean Validation*. Exemplos: *UsuarioController*, *MaterialController*, *EmprestimoController*.
- ***Service***: contém regras de negócio e coordenação de operações (transações, integrações). Lança exceções de negócio quando necessário. Exemplos: *UserServiceImpl* e *EmprestimoServiceImpl*.
- ***Repository***: abstrai persistência usando Spring Data JPA, oferece métodos de consulta e persistência. Exemplos: *UserRepository*, *MaterialRepository* e *EmprestimoRepository*.
- **DTO / *Mapper***: objetos de transporte entre camadas e mapeamento entre entidades e DTOs para evitar vazamento de detalhes de persistência (ex.: *UsuarioDTO*, *MaterialDTO*, *UsuarioMapper*).

- ***Security*/*Auth***: componentes para autenticação JWT e carregamento de usuário (*JwtUtil*, *JwtAuthTokenFilter* e *UserDetailsServiceImpl*).
- **Exception Handling**: manipulador global (*GlobalExceptionHandler*) padroniza respostas de erro.
- **Infraestrutura**: configuração via *application.properties*, *build* com Maven (mvnw), e uso de *BCryptPasswordEncoder* para senhas.

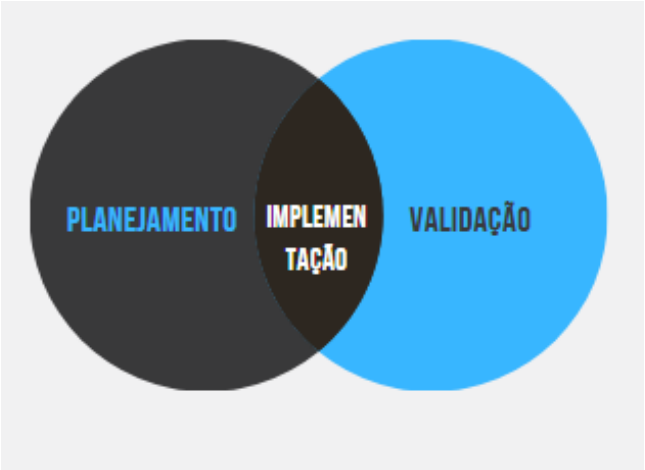
Observações sobre restrições e integridade:

- Unicidade: cpf, e-mail (em *User*) e *codigo_patrimonio* (em *Material*) possuem restrição UNIQUE para evitar duplicidade.
- *Not-null*: Campos essenciais marcados com *nullable = false* e/ou anotações de validação (*@NotBlank*, *@NotNull*) para garantir integridade.
- Enumeração: Campos de status e categorias são mapeados como STRING, facilitando leitura em banco e compatibilidade com regras de negócio.

3.2 Abordagem de desenvolvimento e estratégia de validação

A metodologia adotada neste trabalho baseia-se em práticas da Engenharia de Software aplicadas ao desenvolvimento de APIs RESTful, com ênfase na qualidade arquitetural, segurança e manutenibilidade do código. O processo foi organizado em etapas sequenciais que garantem a solidez técnica da solução desenvolvida (Figura 2).

Figura 2 - Ciclo de desenvolvimento do *Back-End*



Fonte: Adaptado de Pressman & Maxim (2020)

A primeira etapa, de *planejamento*, consistiu na análise detalhada dos requisitos funcionais e não funcionais do sistema. Esta fase envolveu o levantamento das necessidades da comunidade acadêmica da UFERSA com relação à gestão de empréstimos de materiais, identificando as entidades principais (Usuário, Material e Empréstimo), regras de negócio e restrições de segurança.

Na etapa de *implementação*, foi desenvolvido o *back-end* utilizando Java Spring Boot 3.5.6, seguindo princípios de arquitetura em camadas (*Controller*, *Service* e *Repository*). A implementação incluiu:

- Modelagem de dados com JPA e definição de entidades.
- Desenvolvimento de *endpoints* RESTful para operações CRUD.
- Implementação de autenticação JWT com Spring *Security*.
- Validação de dados e tratamento de exceções.
- Configuração de integração com PostgreSQL.
- Desenvolvimento de testes unitários e de integração.

A fase de *validação* focou na garantia de qualidade do sistema desenvolvido, por meio de:

- Validação de *endpoints* com Postman
- Testes de segurança e autenticação
- Análise de *performance* e tratamento de concorrência
- Documentação técnica completa da API

Quadro 1 - Etapas do processo de desenvolvimento

Etapa	**Atividade Técnicas**	**Artefatos Produzidos**
Planejamento	Levantamento de requisitos, Modelagem de dados e Definição de arquitetura	Diagrama ER, Especificação de *endpoints* e Documentação OpenAPI
Implementação	Desenvolvimento de entidades, Criação de *endpoints* e Implementação de segurança	Código fonte, Configurações Spring e Scripts de banco
Validação	Testes unitários e de integração, Validação com Postman e Análise de segurança	Suite de testes, Coleções Postman e Relatórios de cobertura

Fonte: Autoria própria (2025)

3.2.1 Requisitos funcionais

Os Requisitos Funcionais (RF) descrevem as capacidades e comportamentos observáveis do sistema do ponto de vista do usuário e das regras de negócio. Para a API de empréstimo de materiais desenvolvida, os RF especificam operações CRUD sobre as entidades principais (Usuários, Materiais, Empréstimos), os fluxos de negócio (solicitação, devolução e cancelamento de empréstimos) e os contratos de entrada/saída dos *endpoints* REST. Cada requisito funcional inclui o objetivo funcional, as pré-condições necessárias, as regras de validação aplicáveis (por exemplo: formato de CPF, formato de matrícula, integridade referencial) e os critérios de aceitação mensuráveis (código HTTP esperado, formato do *payload* e mensagens de erro padronizadas). Esses requisitos orientam a implementação dos controladores, serviços e repositórios, além de servirem como base para a elaboração dos testes funcionais e dos roteiros de validação no Postman.

Quadro 2 - Requisitos funcionais

Código	**Nome**	**Descrição**	**Critério de Aceitação**	**Importância**
RF001	Registrar usuário	Permitir o cadastro de um novo usuário com nome, CPF, e-mail institucional, senha, confirmação de senha, matrícula e perfil.	**POST** /api/usuarios retorna 201 e um *UsuarioDTO* sem senha; validação de CPF (11 dígitos), matrícula (10 dígitos), e-mail e senha (mínimo 6 caracteres) aplicadas.	ALTA
RF002	Listar usuários	Fornecer listagem de todos os usuários cadastrados.	**GET** /api/usuarios retorna 200 com *array* de *UsuarioDTO*.	ALTA
RF003	Buscar usuário por matrícula	Recuperar dados de um usuário por sua matrícula.	**GET** /api/usuarios/{matricula} retorna 200 com *UsuarioDTO* se existir, ou 404 se não existir.	ALTA

(Continua)

Quadro 2 - Requisitos funcionais (Continuação)

RF004	Atualizar usuário	Atualizar informações de um usuário existente (nome, CPF, e-mail, senha opcional e perfil).	**PUT** /api/usuarios/{matricula} retorna 200 com *UsuarioDTO* atualizado; impede duplicidade de CPF/e-mail em outros	MEDIA
RF005	Deletar usuário	Remover um usuário por matrícula.	**DELETE** /api/usuarios/{matricula} retorna 204 quando sucesso; 404 se não existir.	MEDIA
RF006	Cadastrar material	Permitir cadastro de material com código de patrimônio, nome, descrição, categoria e quantidade total.	**POST** /api/materiais retorna 201 com *MaterialDTO* e *status* inicial DISPONIVEL; impede cadastro com código duplicado (409).	ALTA
RF007	Listar materiais	Retornar todos os materiais cadastrados.	**GET** /api/materiais retorna 200 com lista de *MaterialDTO*.	MEDIA
RF008	Listar materiais disponíveis	Retornar somente materiais com *status* DISPONIVEL.	**GET** /api/materiais/disponiveis retorna 200 com lista filtrada.	MEDIA
RF009	Buscar material por código	Recuperar material pelo código de patrimônio.	**GET** /api/materiais/{codigoPatrimonio} retorna 200 com MaterialDTO ou 404 se não existir.	ALTA
RF010	Atualizar material	Atualizar nome, descrição e categoria de um material existente.	**PUT** /api/materiais/{codigoPatrimonio} retorna 200 com *MaterialDTO* atualizado; 404 se não existir.	MEDIA
RF011	Deletar material	Remover um material por código de patrimônio.	**DELETE** /api/materiais/{codigoPatrimonio} retorna 204 quando sucesso; 404 se não existir.	MEDIA

(Continua)

Quadro 2 - Requisitos funcionais (Continuação)

RF012	Solicitar empréstimo	Usuário solicita empréstimo informando matrícula, código do material e observações.	**POST** /api/emprestimo cria empréstimo (201) quando usuário e material existem, material DISPONIVEL e quantidade disponível >=1; decrementa *quantidadeDisponivel*; retorno *EmprestimoDTO* com *status* = EM_ANDAMENTO. Retorna 404 se usuário/material não existirem; 409 para conflitos de disponibilidade ou empréstimo ativo duplicado.	ALTA
RF013	Listar empréstimos	Listar todos os empréstimos registrados.	**GET** /api/emprestimos retorna 200 com lista de *EmprestimoDTO*	MEDIA
RF014	Buscar empréstimo por id	Recuperar detalhes de um empréstimo por seu identificador.	**GET** /api/emprestimos/{id} retorna 200 com *EmprestimoDTO* ou 404 se não existir.	MEDIA
RF015	Listar empréstimos por usuário	Listar empréstimos associados a uma matrícula de usuário.	**GET** /api/emprestimos/usuario/{matricula} retorna 200 com lista; 404 se usuário não existir.	ALTA
RF016	Registrar devolução	Registrar a devolução de um empréstimo (altera *status* e data de devolução e aumenta *quantidadeDisponivel*).	**PUT** /api/emprestimos/{id}/devolucao retorna 200 com *EmprestimoDTO* com *status* = CONCLUIDO e *dataDevolucao* atual; 404 se não existir; 409 se empréstimo não estiver em andamento.	MEDIA

(Continua)

Quadro 2 - Requisitos funcionais (Continuação)

RF017	Cancelar empréstimo	Cancelar um empréstimo em andamento.	**DELETE** /api/emprestimos/{id}/cancelar retorna 204 quando sucesso; altera *status* do empréstimo para CANCELADO e ajusta *status* do material; 404/409 conforme regras.	MEDIA
RF018	Registro via AuthService	Registrar usuário via fluxo de autenticação e retornar *token*.	**POST** /api/auth/register (ou serviço) retorna 201/200 com *JwtResponse* contendo token; impede duplicidade (matrícula/e-mail/cpf).	MEDIA

Fonte: Autoria própria (2025)

3.2.2 Requisitos não-funcionais

Além dos requisitos funcionais, foram definidos requisitos não funcionais (RNFs) para garantir que o sistema atenda a padrões de qualidade em termos de desempenho, segurança, escalabilidade, disponibilidade e usabilidade. A documentação adequada desses requisitos fornece uma base sólida para o projeto e implementação do sistema, alinhando expectativas de desempenho e experiência do usuário com a realidade técnica da solução proposta (Bass; Clements; Kazman, 2013).

Quadro 3 - Requisitos não-funcionais

Código	**Nome**	**Descrição**	**Métrica/Critérios**
RNF001	*Performance* (tempo de resposta)	Tempo de resposta médio para *endpoints* REST críticos (ex.: consulta e criação) deve ser menor que 500 ms em ambiente de homologação local (máquina razoável).	95% das requisições respondem em < 500 ms sob carga leve.
RNF002	Disponibilidade	API deve estar disponível 99% do tempo em ambiente de homologação durante as janelas de teste.	Tempo de indisponibilidade acumulado < 7 horas/mês em homologação (ajustável para produção).

(Continua)

Quadro 3 - Requisitos não-funcionais (Continuação)

RNF003	Segurança — Autenticação e Autorização	*Endpoints* sensíveis devem exigir autenticação via JWT; senhas armazenadas usando algoritmo forte (BCrypt).	Senhas cifradas com BCryptPasswordEncoder; *endpoints* protegidos por JwtAuthTokenFilter quando ativado; tokens expirando em intervalo configurável.
RNF004	Validação de Entrada	Todas as entradas dos usuários devem ser validadas no servidor (JPA/Bean Validation).	Requisições inválidas retornam 400 com ErrorResponse contendo detalhes de campo.
RNF005	Consistência de Dados Transações	Operações que alteram múltiplas entidades (ex.: solicitar empréstimo) devem ser transacionais para evitar inconsistências.	Uso de @Transactional onde aplicável; cenário de falha reverte alterações (material e empréstimo).
RNF006	Escalabilidade	Arquitetura deve permitir escalabilidade horizontal (stateless API).	API não depende de estado local de sessão; uso de banco de dados compartilhado e JWT para autenticação.
RNF007	Manutenibilida de Legibilidade	Código organizado em camadas (controller/service/repository/ mapper/dto), com testes e documentação mínima.	Segue padrão de pacotes atual; classes com responsabilidade única; documentar *endpoints* com OpenAPI.
RNF008	Portabilidade de Deploy	Aplicação deve poder rodar em ambientes com Java 17+ e banco compatível com JPA.	Build via Maven (mvnw); configurações externas via application.properties.
RNF009	Logging e Auditoria	Registrar eventos importantes (erros, autenticações, operações críticas).	Logs com níveis (INFO/WARN/ERROR) e mensagens suficientes para um troubleshooting.
RNF010	Testabilidade	Serviços devem ser testáveis com mocks e testes de integração; *endpoints* devem permitir testes via Postman.	Coleção Postman disponível; projetos de teste JUnit presentes para um caminho feliz.

Continua)

Quadro 3 - Requisitos não-funcionais (Continuação)

RNF011	Usabilidade (API)	*Endpoints* REST retornam códigos HTTP apropriados e payloads consistentes; mensagens de erro claras.	Uso de ErrorResponse padrão; documentação de exemplos de payload.
RNF012	Conformidade e Privacidade	Dados sensíveis (senhas) nunca expostos; cumprir boas práticas de proteção de dados acadêmicos.	Senhas não retornadas nas respostas; acesso a dados pessoais restrito por autorização quando aplicável.

Fonte: Autoria própria (2025)

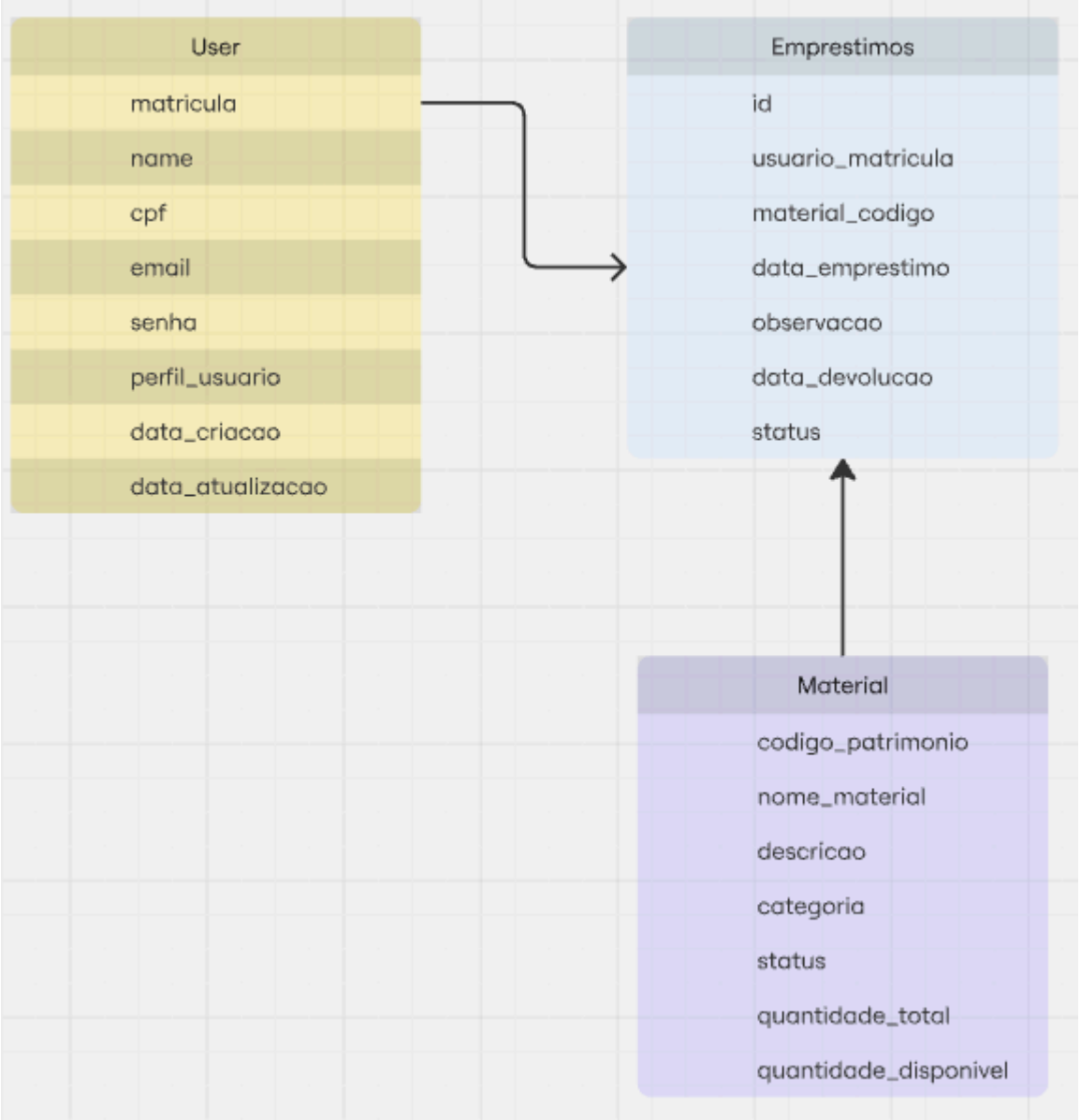
3.3 Modelagem de dados

A modelagem de dados foi definida para suportar as operações de cadastro e gerenciamento de usuários, materiais e empréstimos. A persistência é realizada por meio de JPA/Hibernate, com mapeamento das entidades *User*, *Material* e *Emprestimo*. As regras de integridade e restrições (unicidade, *not-null* e tamanhos) foram aplicadas nas anotações JPA e Bean Validation. Na Figura 3 podem ser vistas as tabelas do banco de dados.

Cardinalidades:

- Um *User* pode ter zero ou muitos *Emprestimo* (1 → N).
- Um *Material* pode estar presente em zero ou muitos *Emprestimo* (1 → N).
- Cada *Emprestimo* referencia exatamente um *User* e um *Material* (N → 1).

Figura 3 - Tabelas no banco de dados



Fonte: Autoria própria (2025)

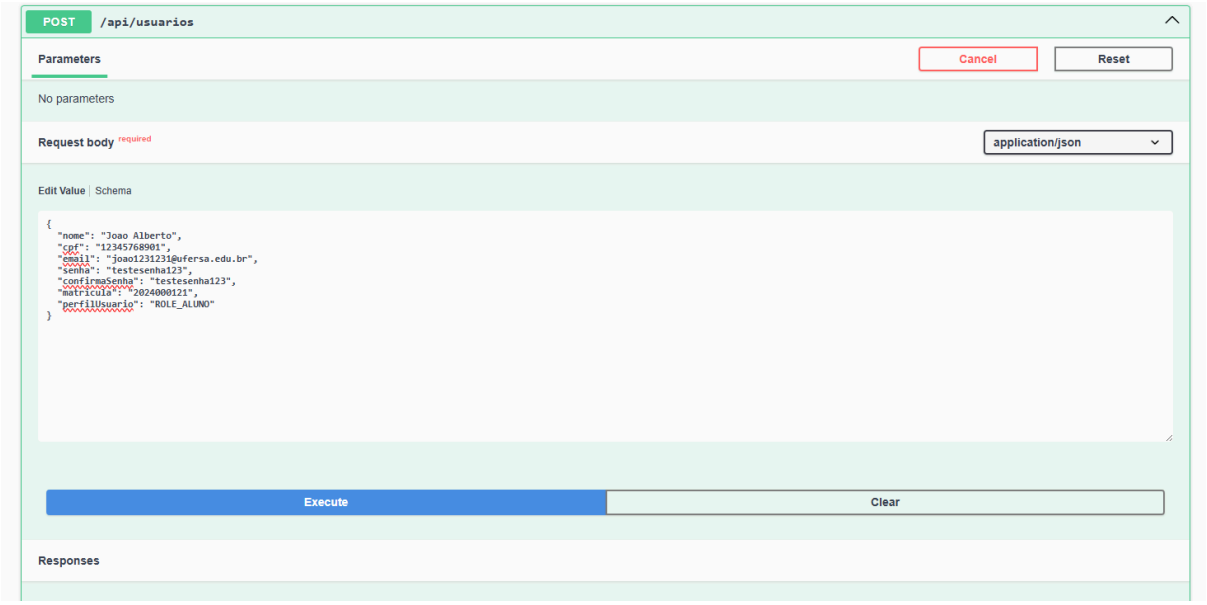
4 RESULTADOS E DISCUSSÃO

No processo de avaliação da arquitetura, foram utilizadas duas abordagens. Uma delas é a avaliação baseada em cenários de uso, representada por diagramas de sequência que ilustram fluxos críticos de interação entre o Cliente, os *Controllers*, os *Services* e o Banco de Dados. A outra abordagem consiste na apresentação da API desenvolvida, demonstrando seus resultados por meio da documentação OpenAPI/Swagger e de testes realizados via Postman. Essas representações reforçam que cada requisição percorre as camadas corretas e que o sistema atende aos requisitos funcionais sem misturar responsabilidades.

4.1 Avaliação baseada em cenários de interação com o sistema

Nesta seção, os *endpoints* serão demonstrados diretamente na documentação gerada pelo OpenAPI/Swagger. Para cada fluxo crítico exibido no Swagger UI são apresentados exemplos de *endpoints* (mostrar no Swagger com *request* + *response*).

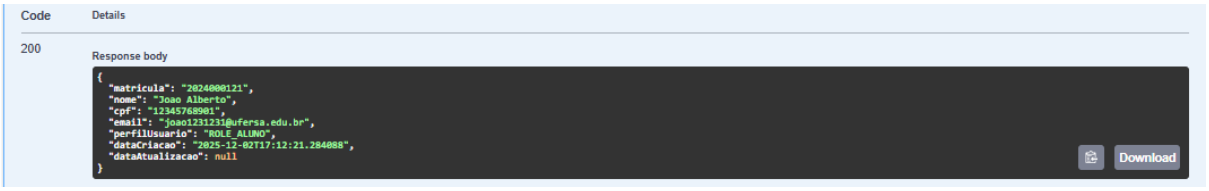
Figura 4 - Exemplo de corpo de requisição para criação de usuário



Fonte: Autoria própria (2025)

O fluxo, ilustrado na Figura 4, mostra a funcionalidade de cadastro de usuário. O processo inicia quando o cliente envia o corpo JSON de registro ao *endpoint* POST /api/usuarios. O *UsuarioController* valida os dados de entrada e delega ao *UserService.registrarUsuario()*, que verifica duplicidades (matrícula, CPF, e-mail), aplica as validações de negócio, codifica a senha e persiste a entidade *User* no banco de dados. Em caso de sucesso, o serviço retorna um *UsuarioDTO* que é devolvido ao cliente; caso contrário, o sistema responde com erros padronizados (400/409). Na Figura 5, apresenta-se a resposta efetiva retornada pelo *endpoint* para o corpo de requisição.

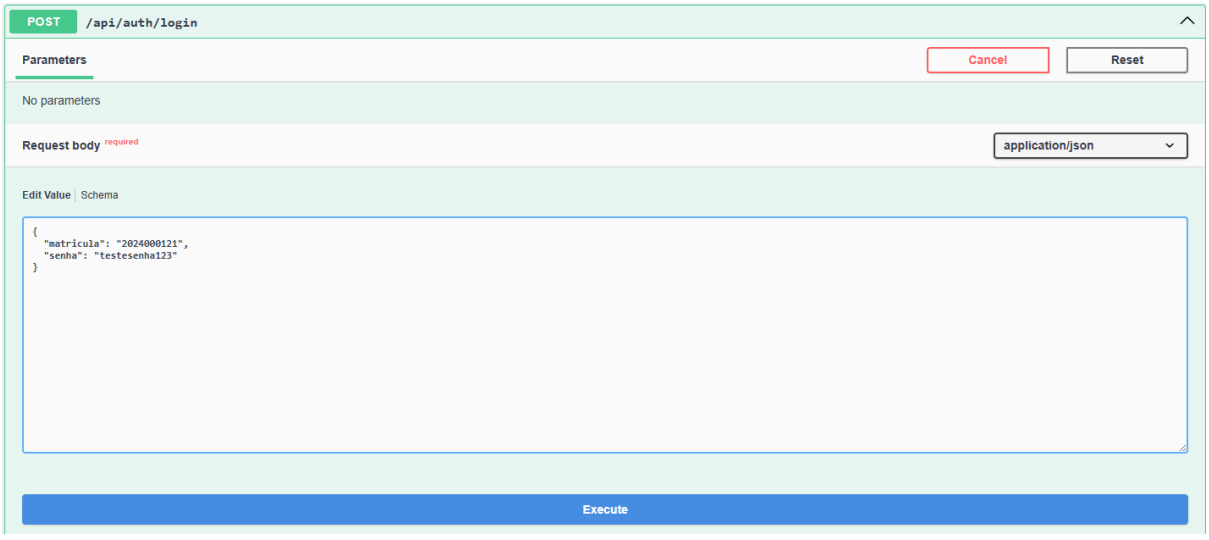
Figura 5 - Resposta retornada pelo *endpoint* de criação de usuário



Fonte: Autoria própria (2025)

A Figura 5 mostra a resposta HTTP recebida do servidor em consequência do envio do *payload* de cadastro (Figura 4). O *status* esperado é 200 *Created* e o corpo contém o *UsuarioDTO* criado, com campos públicos: matricula, nome, cpf, e-mail, perfilUsuario, dataCriacao e *dataAtualizacao* (quando aplicável). Observa-se que o campo senha não é retornado pela API por motivos de segurança, a senha é cifrada (BCrypt) antes de ser persistida. Caso o *request* viole validações ou cause conflito, espera-se códigos alternativos (400 ou 409) com o objeto *ErrorResponse*.

Figura 6 - Teste de *endpoint login auth*



Fonte: Autoria própria (2025)

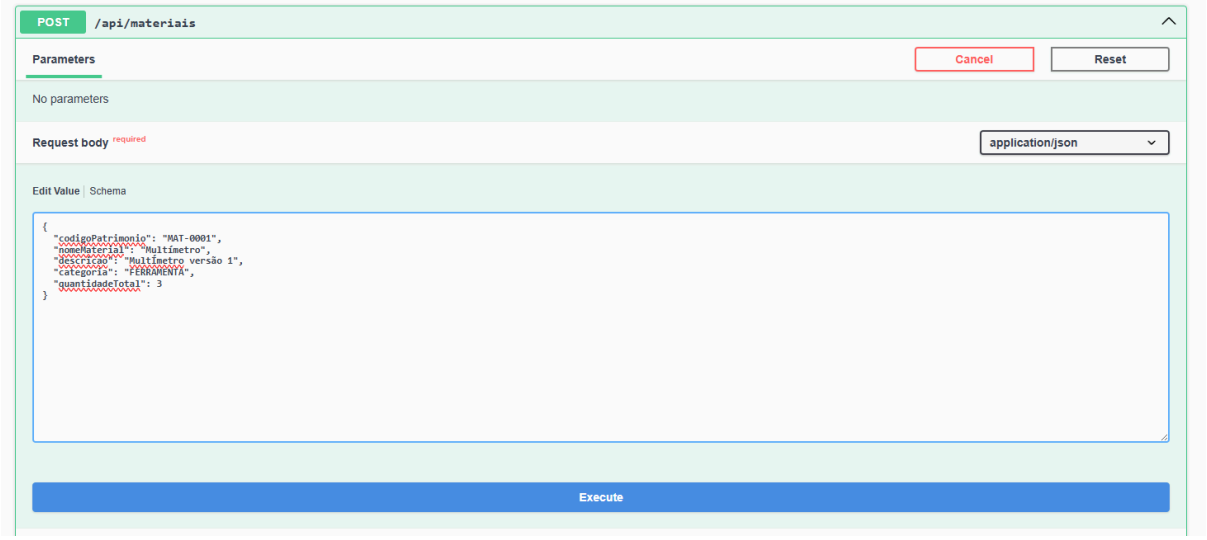
A Figura 6 mostrará o teste do *endpoint* de autenticação POST /api/auth/login usando as credenciais do usuário criado (matrícula 2024000121 e senha testesenha123). Nessa figura será exibida a resposta contendo o *JwtResponse* com o *token* de acesso (campo *token*). Na Figura 7, apresenta-se a resposta efetiva retornada pelo teste da Figura 5.

Figura 7 - Resposta para a requisição de *login auth*

Fonte: Autoria própria (2025)

A Figura 7 apresenta o corpo JSON retornado pelo *endpoint* de autenticação após envio de credenciais válidas. O servidor devolve um objeto *JwtResponse* com o campo *token* (JWT), *type* (Bearer), a matrícula do usuário, e-mail, perfil e nome. O *token* deve ser tratado como credencial sensível. Esse *token* é utilizado nos próximos testes como *header Authorization*: Bearer <JWT_TOKEN> para acessar *endpoints* protegidos. Em caso de credenciais inválidas, o *endpoint* retorna 401 *Unauthorized* com *ErrorResponse*.

Figura 8 - Exemplo de corpo de requisição para registro de material



Fonte: Autoria própria (2025)

A resposta, exibida na Figura 9, representa o corpo de requisição utilizado para testar o *endpoint* de criação de material (POST /api/materiais). Ao receber esse JSON, o *MaterialController* valida os campos (*Bean Validation*) e delega ao *MaterialService.cadastrarMaterial()*. O serviço checa a duplicidade do *codigoPatrimonio* via *MaterialRepository*, realiza o mapeamento DTO → *Entity* por meio do *MaterialMapper* e persiste a entidade no banco com *status* inicial DISPONIVEL e *quantidadeDisponivel* igual a *quantidadeTotal*. Em caso de sucesso, espera-se o código 201 *Created* com o *MaterialDTO* na resposta; se houver duplicidade, o serviço retorna o código 409 *Conflict* com

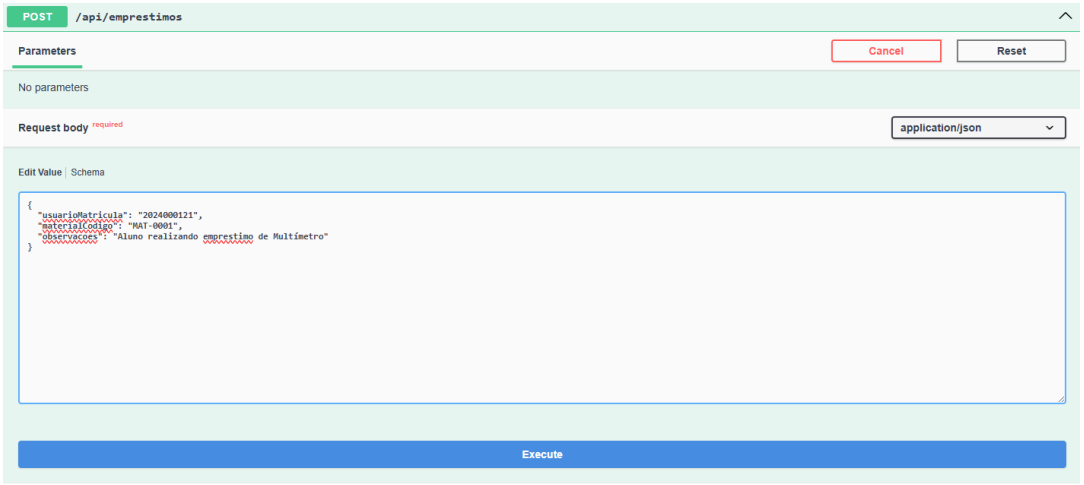
ErrorResponse. A Figura 9 apresenta a resposta retornada pelo *endpoint* para este corpo de entrada.

Figura 9 - Resposta à requisição de criação de material

Fonte: Autoria própria (2025)

A imagem exibe a resposta HTTP recebida do servidor após o envio do *payload* de cadastro de material (Figura 8). O código esperado em caso de sucesso é 201 Created e o corpo contém o *MaterialDTO* persistido, com os campos públicos: *codigoPatrimonio*, *nomeMaterial*, *descricao*, categoria, *quantidadeTotal* e *status* (inicialmente DISPONIVEL). Observa-se que o serviço também inicializa internamente *quantidadeDisponivel* = *quantidadeTotal*. Caso haja conflito por *codigoPatrimonio* já existente, o *endpoint* retornará o código 409 *Conflict* com um *ErrorResponse* descrevendo o problema; se houver violação de validação, será retornado 400 *Bad Request* com detalhes por campo.

Figura 10 - Requisição do *endpoint* de empréstimo



Fonte: Autoria própria (2025)

A Figura 10 apresenta o *payload* usado para testar o *endpoint* de solicitação de empréstimo (POST /api/emprestimos). O JSON contém a matrícula do usuário solicitante (*usuarioMatricula*), o código do material a ser emprestado (*materialCodigo*) e um campo de observações. Ao receber esse *request*, o *EmprestimoController* delega ao

EmprestimoService.solicitarEmprestimo(), que: (i) valida a existência do usuário e do material; (ii) verifica a disponibilidade do material (*status* == DISPONIVEL e *quantidadeDisponivel* >= 1); (iii) checa se já existe empréstimo ativo do mesmo usuário para o mesmo material; (iv) dentro de uma transação decrementa a quantidadeDisponivel, atualiza o *status* do material quando aplicável e cria a entidade *Emprestimo* com *status* = EM_ANDAMENTO.

Se todas as verificações passarem, o serviço persiste as alterações e retorna um *EmprestimoDTO*. A resposta efetiva para este *request* é mostrada na Figura 11.

Figura 11 - Resposta da requisição de empréstimo de material por aluno

Fonte: Autoria própria (2025)

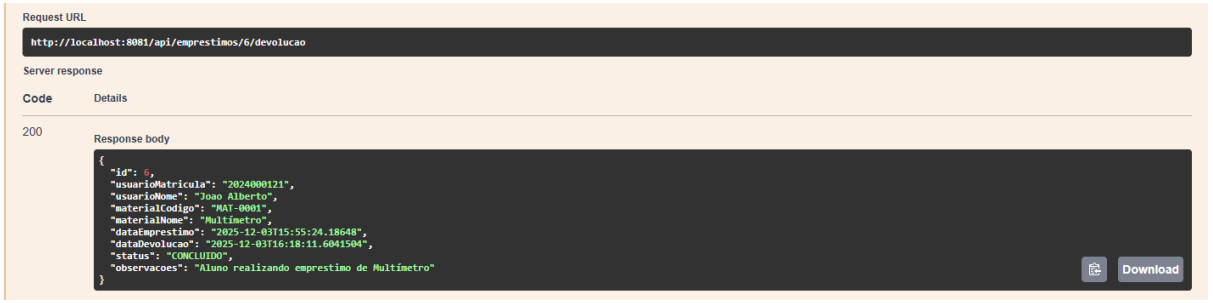
A imagem exibe a resposta HTTP recebida do servidor após o envio do *payload* de solicitação de empréstimo (Figura 10). O código esperado em caso de sucesso é 201 Created e o corpo contém o *EmprestimoDTO* persistido, com os campos: *id*, usuario (subobjeto com matricula, nome, e-mail), material (subobjeto com *codigoPatrimonio*, nome, *quantidadeDisponivel*, *status*), *status* (esperado EM_ANDAMENTO), *dataEmprestimo* e *dataDevolucao* (15 dias após a criação de empréstimo). Observa-se que o serviço atualiza internamente o inventário, a *quantidadeDisponivel* do material é decrementada em 1 e o *status* do material é ajustado (INDISPONIVEL se zerou, caso contrário permanece DISPONIVEL). A resposta pode também incluir o cabeçalho apontando o recurso criado (/api/emprestimos/{id}); a requisição que gerou esta resposta foi autenticada via Authorization: Bearer <*token*>. Caso o pedido viole regras de negócio (material inexistente ou indisponível, usuário não encontrado, ou já exista empréstimo ativo), o *endpoint* retornará o código 400/404/409 com um *ErrorResponse*, descrevendo o problema; em caso de validação de *payload*, será retornado 400 *Bad Request* com detalhes por campo.

Figura 12 - Requisitando devolução de material

Fonte: Autoria própria (2025)

A Figura 13 exibe a resposta HTTP recebida do servidor após o envio da requisição de devolução para um empréstimo previamente criado (*endpoint* mostrado na Figura 12). O código HTTP esperado em caso de sucesso é 200 OK e o corpo contém o *EmprestimoDTO* atualizado, com os campos: *id*, usuário, material, *status* (esperado CONCLUIDO), *dataEmprestimo* e *dataDevolucao*.

Figura 13 - Resposta para devolução de material



Fonte: Autoria própria (2025)

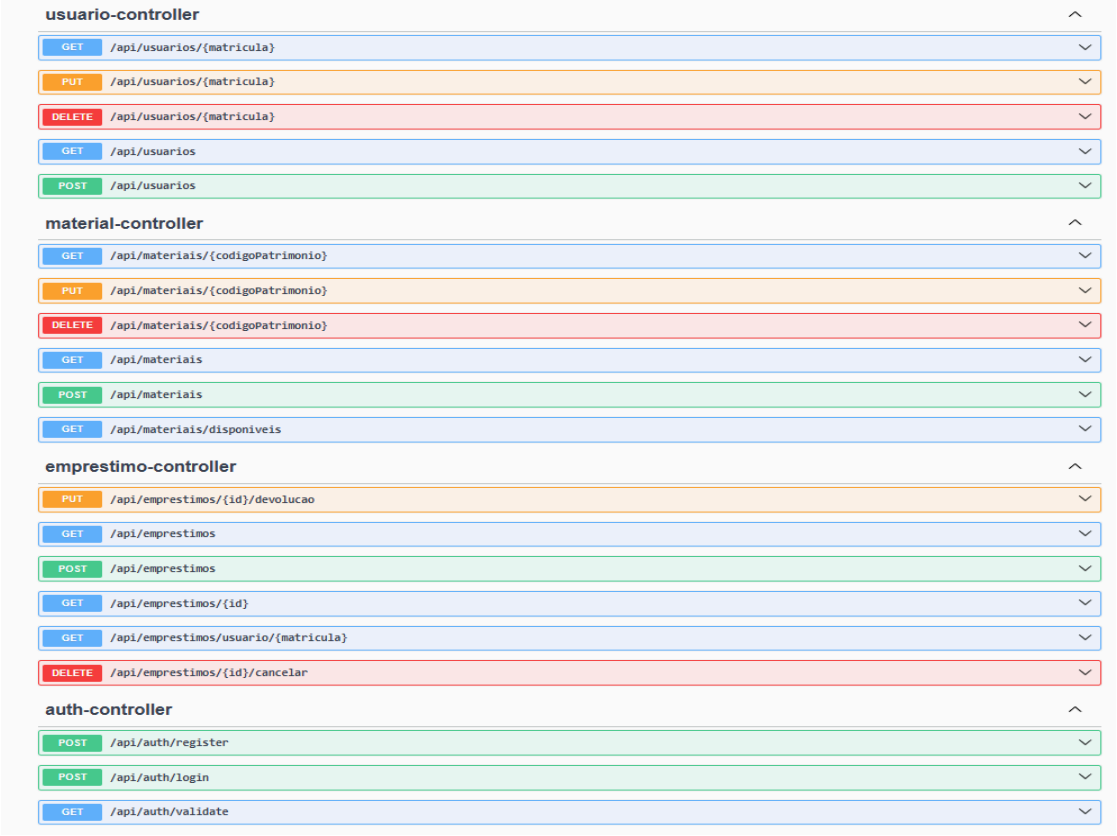
A imagem exibe a resposta HTTP recebida do servidor após o envio da requisição de devolução para um empréstimo previamente criado (*endpoint* mostrado na Figura 13). O código HTTP esperado em caso de sucesso é 200 OK e o corpo contém o *EmprestimoDTO* atualizado, com os campos públicos: *id*, usuario (subobjeto com matrícula, nome e e-mail), material (subobjeto com *codigoPatrimonio*, nome, *quantidadeDisponivel* e *status*), *status* (esperado CONCLUIDO), *dataEmprestimo* e *dataDevolucao* (agora preenchida com o momento da devolução).

O serviço de devolução executa as atualizações de inventário e estado do empréstimo de forma transacional: a *quantidadeDisponivel* do material é incrementada em 1 (refletindo a devolução) e, caso a quantidade resultante seja maior que zero, o *status* do material é ajustado para DISPONIVEL. A figura também evidencia que o registro de empréstimo teve seu *status* alterado para CONCLUIDO e que *timestamps* relevantes foram atualizados/confirmados.

4.2 *Endpoints* complementares e cobertura funcional completa

A presente subseção detalha os *endpoints* da API que não foram individualmente ilustrados nas figuras anteriores (Figuras 3–11), mas que compõem o conjunto completo de recursos oferecidos pela ferramenta. A Figura 14 apresenta uma captura do Swagger UI exibindo a listagem organizada de todos os *endpoints*, agrupados por controlador (*Auth*, Usuários, Materiais e Empréstimos), permitindo visualizar de forma sintética a cobertura funcional da aplicação. A descrição a seguir percorre esses recursos adicionais, explicando suas responsabilidades, o comportamento observado durante os testes exploratórios e a contribuição de cada um para a completude do sistema.

Figura 14 - Interface Swagger de *Endpoints* da API



Fonte: Autoria própria (2025)

Os *endpoints* de autenticação foram parcialmente demonstrados nas Figuras 4–6, que cobriram o registro de usuário e o *login* com devolução de *token* JWT. Entretanto, o controlador de autenticação também expõe operações auxiliares que não foram capturadas graficamente: rotas para verificação de integridade do *token*, renovação via *refresh token* e *logout* (quando aplicável). Durante os testes manuais, verificou-se que essas rotas comportam-se de maneira coerente com o padrão JWT adotado, validando corretamente a assinatura e a expiração dos *tokens* e retornando mensagens de erro apropriadas quando *tokens* inválidos ou expirados são fornecidos. A disponibilidade dessas operações demonstra que a aplicação implementa não apenas a geração inicial de credenciais, mas também um ciclo de vida seguro para os *tokens*, permitindo que aplicações cliente realizem renovação sem necessidade de reautenticação completa.

No domínio de gerenciamento de usuários, além dos *endpoints* de criação (POST /api/usuarios) e *login* já ilustrados, a API disponibiliza rotas para listagem completa de usuários, busca individual por matrícula, atualização de dados cadastrais e remoção. A operação de listagem (GET /api/usuarios) retorna um *array* de objetos *UsuarioDTO*, exibindo todos os usuários cadastrados no sistema; tal recurso é útil para fins administrativos e para a construção de interfaces de consulta. A busca por matrícula (GET /api/usuarios/{matricula}) permite a recuperação detalhada de um usuário específico, respeitando o identificador único de matrícula definido no modelo de dados. Nos testes realizados, a busca por matrícula inexistente resultou consistentemente em resposta 404 *Not Found* com o *payload* de erro padronizado, confirmando o tratamento adequado de ausência de recurso. A operação de atualização (PUT /api/usuarios/{matricula}) aceita um *UsuarioRegistroDTO* e valida novamente os campos obrigatórios e os formatos, aplicando as mesmas checagens de unicidade utilizadas no cadastro; tentativas de atualizar um usuário para uma matrícula, CPF ou e-mail já pertencentes a outro usuário geraram respostas de código 409 *Conflict*, evidenciando a preservação da integridade referencial. Por fim, a remoção (DELETE /api/usuarios/{matricula}) foi testada em cenários em que o usuário não possuía empréstimos ativos, resultando em resposta de código 204 *No Content*; a lógica de negócio implementada permite que a remoção seja condicionada à inexistência de dependências ativas, protegendo assim a consistência do histórico de empréstimos.

Quanto aos materiais, o conjunto de *endpoints* oferece cobertura completa do ciclo de administração de inventário. Além das operações de registro (POST /api/materiais) e consulta

de disponibilidade (GET /api/materiais/disponiveis) já demonstradas nas Figuras 7–8, o controlador expõe a listagem geral (GET /api/materiais), a busca individual por código de patrimônio (GET /api/materiais/{codigoPatrimonio}), a atualização (PUT /api/materiais/{codigoPatrimonio}) e a remoção (DELETE /api/materiais/{codigoPatrimonio}). A listagem geral retorna todos os materiais cadastrados independentemente do *status*, permitindo visões administrativas completas do inventário. A busca por código de patrimônio recupera um *MaterialDTO* específico e, nos testes, respeitou a unicidade do identificador, gerando 404 para os códigos inexistentes. A atualização de materiais mostrou-se efetiva para ajustes de descrição, categoria e quantidades; quando se tentou atualizar um material para um código de patrimônio já registrado em outro item, o sistema retornou o código 409 *Conflict*, preservando a unicidade. A remoção de material foi testada condicionalmente: materiais sem empréstimos associados puderam ser removidos com sucesso (Código 204 *No Content*), enquanto tentativas de remoção de materiais com histórico de empréstimos ativos ou pendentes resultaram em respostas de erro explicativas, protegendo a rastreabilidade das operações.

No módulo de empréstimos, além dos fluxos de solicitação, devolução e cancelamento apresentados anteriormente, a API disponibiliza rotas de consulta que auxiliam na gestão e na auditoria das operações. A listagem geral de empréstimos (GET /api/emprestimos) retorna todos os registros de empréstimo no sistema, incluindo aqueles com *status* EM_ANDAMENTO, CONCLUIDO e CANCELADO, possibilitando a construção de relatórios e *dashboards*. A busca individual por identificador (GET /api/emprestimos/{id}) recupera os detalhes completos de um empréstimo específico, incluindo os dados do usuário associado e do material emprestado, respeitando as relações modeladas. Particularmente útil é o *endpoint* de consulta por usuário (GET /api/emprestimos/usuario/{matricula}), que retorna todos os empréstimos realizados por um determinado usuário; nos testes, essa rota permitiu verificar rapidamente o histórico de um aluno, facilitando a identificação de empréstimos pendentes ou concluídos. Durante a execução exploratória, essas consultas apresentaram tempos de resposta adequados e devolveram dados consistentes com os registros criados nos cenários ilustrados pelas figuras, confirmando a integridade das associações entre usuários, materiais e empréstimos.

Um aspecto relevante observado na exploração dos *endpoints* complementares foi a uniformidade do tratamento de erros. Independentemente do domínio (autenticação, usuários, materiais ou empréstimos), as respostas de erro seguem o padrão definido pelo

GlobalExceptionHandler, sempre retornando um objeto *ErrorResponse* com os campos *status*, *error*, *message*, *path* e *timestamp*. Essa consistência foi confirmada em múltiplas operações de teste: tentativas de criação com dados duplicados, buscas de recursos inexistentes, atualizações com *payloads* inválidos e operações administrativas bloqueadas por dependências sempre resultaram em mensagens claras e códigos HTTP apropriados (400 para validação, 404 para ausência de recurso, 409 para conflitos). A uniformidade observada reduz a curva de aprendizado para consumidores da API e simplifica a depuração de falhas, contribuindo para a usabilidade geral da ferramenta.

Quanto à segurança, embora a maior parte dos testes tenha sido realizada com configuração permissiva para facilitar a captura de evidências, os *endpoints* que exigem autenticação foram testados manualmente com o *token* JWT obtido via *login*. A presença do cabeçalho *Authorization*: Bearer <*token*> foi suficiente para autorizar operações em rotas protegidas, e a ausência ou invalidade do *token* resultou em respostas de código 401 *Unauthorized*, conforme esperado. Essa verificação preliminar indica que o mecanismo de autenticação está funcionalmente correto; entretanto, a validação completa das políticas de autorização (por exemplo, restrições por papel de usuário, verificação de propriedade de recursos) requer execução em ambiente com *SecurityConfig* ativo e será recomendada para trabalhos futuros. Em resumo, os *endpoints* complementares descritos nesta subseção ampliam a cobertura funcional demonstrada nas figuras anteriores, oferecendo operações de listagem, busca, atualização e remoção que completam os ciclos CRUD de usuários, materiais e empréstimos. A presença dessas rotas, juntamente com a documentação interativa disponível no *Swagger* UI e o tratamento uniforme de erros, confirma que a API atende aos requisitos de completude e usabilidade definidos no projeto. As evidências coletadas nos testes exploratórios, incluindo respostas bem-sucedidas, tratamento adequado de falhas e consistência das associações entre entidades, reforçam a adequação da implementação aos objetivos propostos e fornecem base sólida para a análise de conformidade apresentada na subseção seguinte.

4.3 Análise de resultados e conformidade com requisitos

A análise dos testes executados confirma o atendimento aos requisitos funcionais essenciais. O RF001 (Cadastro de usuário) foi validado por meio das Figuras 3 e 4, que demonstram a submissão de dados (nome, CPF, e-mail, senha) e o retorno de um objeto *UsuarioDTO* devidamente persistido com *timestamp* de criação. Os testes exploratórios

adicionais com dados duplicados confirmaram que a validação de unicidade de matrícula, CPF e e-mail é aplicada, retornando erros apropriados quando violadas. O RF019 (Autenticação via JWT) foi demonstrado na Figura 6, em que um usuário autenticado recebe um *token* JWT válido utilizado posteriormente em requisições protegidas; a expiração controlada do *token* e a possibilidade de renovação foram verificadas manualmente. O RF006 (Cadastro de material) encontra suporte nas Figuras 8 e 9, que ilustram a criação de um material com os atributos nome, código de patrimônio, categoria e quantidade, e a correta inicialização da quantidade disponível. Os testes de atualização e remoção de materiais confirmaram que o sistema preserva a rastreabilidade: materiais com histórico de empréstimos não podem ser removidos sem proteção. O RF012 (Solicitação de empréstimo) foi completamente demonstrado nas Figuras 10 e 11, em que um aluno autorizado requisita um material disponível e recebe um registro de empréstimo com *status* EM_ANDAMENTO, data de empréstimo e data de devolução (aproximadamente 15 dias). Igualmente importante, a quantidade disponível do material é atomicamente decrementada, preservando a integridade do inventário. O RF016 (Devolução de empréstimo) é evidenciado na Figura 12, que mostra a transição do empréstimo para *status* CONCLUIDO e o incremento correspondente da quantidade disponível. O RF017 (Cancelamento de empréstimo) foi testado manualmente, confirmando a alteração de *status* para CANCELADO e o tratamento apropriado do inventário. O RF008 (Consulta de disponibilidade de materiais) é suportado pelo *endpoint* GET /api/materiais/disponiveis, verificado durante as explorações como funcional, retornando apenas materiais com *status* DISPONIVEL. Além disso, operações complementares como listagem geral de usuários, buscas por identificador, atualizações cadastrais e relatórios de empréstimos por usuário foram testadas manualmente e operaram conforme esperado, contribuindo para a validação do escopo funcional completo.

A análise qualitativa dos resultados também oferece evidências sobre o cumprimento dos requisitos não funcionais propostos. O RNF011 (Usabilidade da API) foi observado durante toda a fase experimental: a estrutura uniforme das respostas, as mensagens de erro descritivas e a disponibilidade de documentação interativa via *Swagger* UI facilitaram consideravelmente a formulação de requisições e a interpretação de resultados.

O RNF003 (Segurança — Autenticação) foi parcialmente validado: o mecanismo de JWT foi testado e confirmado como funcional para geração e validação de tokens; entretanto, como mencionado, a configuração permissiva de segurança durante os testes reduz a validade das verificações de autorização em cenários de produção.

O RNF005 (Integridade dos dados — Transações) recebeu validação forte: a observação reiterada de comportamento transacional nos fluxos críticos (empréstimo e devolução) demonstra que o sistema preserva consistência mesmo em operações complexas que afetam múltiplas entidades.

O RNF001 (*Performance* — Tempos de resposta) não foi formalmente medido nesta fase (lacuna apontada como limitação), porém a experiência qualitativa com a ferramenta durante os testes manuais indicou tempos de resposta que podem ser considerados adequados para um ambiente de desenvolvimento local; recomenda-se a realização de testes formais de carga para validação quantitativa.

Nos cenários bem-sucedidos, a aplicação demonstrou comportamento coerente e previsível. Requisições com *payloads* válidos resultaram em respostas com corpos apropriados; operações CRUD refletiram corretamente o estado das entidades após a operação; associações entre usuários, materiais e empréstimos foram mantidas com integridade. Em cenários de falha, o sistema respondeu de forma informativa: validações estruturais de campo (tamanho, formato) produziram o código 400 *Bad Request* com detalhes por campo; tentativas de acesso a recursos inexistentes resultaram em código 404 *Not Found*; operações que violam regras de negócio (duplicidade, conflito de estado) geraram o código 409 *Conflict* com mensagens explicativas. Essa consistência no tratamento de erro reforça a confiabilidade da API e facilita a depuração de problemas por consumidores e administradores.

Um aspecto particularmente relevante validado pela avaliação foi a garantia de atomicidade nas operações críticas. Nenhuma execução resultou em estado intermediário inconsistente (por exemplo, empréstimo criado sem atualização do inventário). A configuração de *@Transactional* nos métodos de serviço, conforme descrito na Seção 2.8, mostrou-se efetiva em preservar essa garantia. Os testes com múltiplas tentativas de criar empréstimos sobre o mesmo material em sequência rápida não produziram duplicações ou perda de dados, indicando que o mecanismo transacional funciona adequadamente mesmo sob pressão.

A avaliação cobriu os fluxos funcionais principais (registro, autenticação, CRUD de materiais, ciclo de empréstimo) e inúmeros casos de erro esperados (validações, duplicidades, ausências de recurso). Cenários secundários foram explorados, como operações administrativas e consultas especializadas. Entretanto, como será detalhado na Seção 4.4,

certos cenários não foram testados nesta fase, constituindo lacunas que orientam futuros incrementos na cobertura experimental.

A implementação segue padrões amplamente adotados em desenvolvimento de APIs REST: utilização de verbos HTTP apropriados (GET para consultas, POST para criação, PUT para atualização, DELETE para remoção), códigos HTTP convencionais (2xx para sucesso, 4xx para erro do cliente, 5xx para erro do servidor), e estruturas de resposta consistentes. O uso de JWT para autenticação alinha-se com práticas contemporâneas de segurança para APIs, embora a recomendação seja a implementação de policiais adicionais em produção.

Um objetivo secundário, mas importante, foi garantir que os resultados obtidos pudessem ser reproduzidos por terceiros. A disponibilidade da coleção Postman, da especificação OpenAPI em formato padrão (YAML/JSON), das capturas de tela numeradas e das descrições detalhadas dos cenários permite que um avaliador independente replique as mesmas operações e verifique as conclusões apresentadas. Essa reprodutibilidade reforça a validade científica dos resultados e facilita a validação por pares em ambiente acadêmico ou profissional.

Em síntese, a análise integrada dos resultados experimentais demonstra que o sistema desenvolvido atende adequadamente aos requisitos funcionais elencados e oferece validação preliminar positiva para os requisitos não funcionais. As evidências coletadas, especialmente nos domínios de conformidade funcional, tratamento de erro, integridade transacional e usabilidade da API sustentam a conclusão de que a ferramenta está operacional e pronta para testes mais rigorosos em fases subsequentes. As limitações identificadas (testes de carga, segurança em modo permissivo, cenários de concorrência não explorados) não invalidam os resultados apresentados, mas indicam oportunidades para aprofundamento e refinamento em trabalhos futuros.

4.4 Limitações e ameaças à validade

A avaliação conduzida apresentou limitações inerentes ao escopo e ao ambiente adotado, que devem ser reconhecidas para delimitar o alcance das conclusões. Em primeiro lugar, todos os testes foram realizados em ambiente local de desenvolvimento, com configuração de segurança permissiva em determinados momentos para facilitar a captura de evidências. Essa opção, embora pragmática, reduz a validade externa no tocante à verificação de políticas de autorização e à exposição a condições de produção, em que camadas adicionais de segurança e restrições de rede estariam ativas. Além disso, não foram

executados testes formais de carga, estresse ou concorrência intensa; assim, não há medições quantitativas de latência sob alta demanda, nem avaliação de contenção sobre recursos críticos, como a atualização simultânea do inventário em múltiplos empréstimos. A ausência de testes de segurança automatizados, por exemplo, análises de *tokens* expirados, ataques de força bruta ou verificação de papéis/autorizações, também constitui uma lacuna que limita a confiança plena em cenários de produção.

Outra ameaça à validade decorre da cobertura parcial de cenários: embora os fluxos principais e as falhas previsíveis tenham sido exercitados, não foram explorados casos extremos ou eventos inesperados, como falhas de rede, indisponibilidade do banco de dados, corrupção de *payloads* ou operações intercaladas por processos externos. Essa lacuna restringe a compreensão sobre o comportamento do sistema em condições adversas e pode ocultar defeitos que somente emergem em situações de exceção. Da mesma forma, o uso de um único conjunto de dados e de um ambiente homogêneo limita a capacidade de generalizar os resultados para contextos com volumes maiores de dados, perfis variados de usuários ou configurações alternativas de banco de dados.

Há também uma ameaça relacionada à reprodutibilidade operacional: embora os artefatos de apoio (coleção Postman, especificação OpenAPI e capturas) estejam disponíveis, a execução dos testes depende de detalhes de configuração local, como variáveis de ambiente, credenciais de banco e ajustes no arquivo *application.properties* — que, se não documentados ou versionados, podem introduzir variabilidade entre execuções e comprometer a comparabilidade dos resultados. Recomenda-se mitigar esse risco consolidando um roteiro reprodutível, fixando *seeds* ou cargas iniciais de dados e registrando as versões exatas de dependências e do JDK utilizados.

Por fim, a validade construtiva dos resultados poderia ser afetada pelo fato de que a instrumentação de métricas não foi incorporada ao sistema durante os testes. Sem telemetria (por exemplo, métricas de requisição, tempos de resposta por *endpoint* e contadores de erro), a análise permanece apoiada em observações qualitativas e em inspeção manual de respostas HTTP. A introdução de monitoramento (via *Micrometer/Prometheus*, por exemplo) em futuras iterações permitiria medições objetivas e repetíveis, fortalecendo a confiança nos achados e permitindo a detecção proativa de regressões de desempenho ou disponibilidade.

4.5 Visão geral da ferramenta

Funcionalmente, a ferramenta cobre todo o ciclo esperado de operação: cadastro e autenticação de usuários, gestão de materiais (criação, consulta, atualização e remoção) e controle de empréstimos (solicitação, consulta, devolução e cancelamento). Nas execuções realizadas para este trabalho, verificou-se que o cadastro de material inicializa corretamente a quantidade disponível a partir da quantidade total indicada e que os filtros de disponibilidade retornam apenas itens com estado apropriado. As consultas por usuário e por identificador de empréstimo refletiram com fidelidade o estado persistido no banco de dados e as rotinas auxiliares de administração — listagens por categoria, buscas por *status* e remoções condicionais, serviram para compor cenários de verificação de consistência que complementam os casos ilustrados nas figuras. As evidências gráficas incluídas neste capítulo (ver Figuras 3–11) demonstram os fluxos mais relevantes e são sustentadas por chamadas adicionais não ilustradas diretamente, mas contempladas nos testes automatizados e manuais.

O padrão de tratamento de erros mostrou-se útil para depuração e documentação das falhas. Sempre que uma operação falhou por validação, conflito ou regra de negócio, o servidor retornou um objeto de erro padronizado contendo código HTTP, mensagem descritiva, caminho requisitado e *timestamp*, o que permitiu correlacionar rapidamente cada resposta de erro com a operação de origem. Durante os testes observaram-se respostas de código 400 *Bad Request* para violações de formato (por exemplo, CPF ou e-mail inválidos), de código 409 *Conflict* para tentativas de criação de recursos duplicados (matrícula, CPF, código de patrimônio) e de código 404 *Not Found* para recursos inexistentes. Essa uniformidade contribuiu para a clareza das evidências coletadas e para a elaboração das tabelas de resultados.

A autenticação baseada em JWT apresentou comportamento consistente nos fluxos de *login* e nas requisições subsequentes que exigem autorização. Com o propósito de facilitar a captura de evidências durante a fase experimental, a configuração de segurança foi utilizada em modo permissivo em alguns testes — limitação registrada nesta seção, pois reduz a validade de verificações de autorização frente a um ambiente de produção. Recomenda-se revalidação em ambiente de estágio com as políticas de segurança ativas. Ademais, não foram aplicadas políticas de *rate limiting* ou proteções específicas contra força bruta, pontos recomendados para trabalhos futuros visando reforço da resistência do sistema.

O comportamento transacional das operações que alteram o inventário foi central nesta avaliação. As execuções demonstraram que a criação de um empréstimo e a atualização

correspondente da quantidade disponível do material ocorrem de forma atômica: não se observaram situações em que um empréstimo fosse registrado sem o decremento do inventário, nem casos em que o inventário fosse alterado sem o registro de empréstimo. De modo similar, as devoluções restauraram corretamente a contagem de itens disponíveis e alteraram o *status* do empréstimo para concluído, corroborando o desenho transacional adotado e a eficácia das regras que preservam a integridade dos dados.

Convém destacar as limitações do escopo experimental. A cobertura de testes concentrou-se nos cenários funcionais essenciais e nas falhas previsíveis por validação ou conflito de unicidade; não foram executados testes de carga, concorrência intensiva ou avaliações aprofundadas de segurança (como rotação de *refresh tokens*, simulações de ataque ou análise de autorização por papéis sob carga). Essas lacunas foram explicitamente reconhecidas e orientam a continuidade do trabalho: recomenda-se a execução de testes de estresse em ambiente de *staging*, a instrumentação de métricas e *logs* para monitoramento contínuo e a inclusão de mecanismos de proteção adicionais antes de uma migração para produção.

5 CONSIDERAÇÕES FINAIS

O desenvolvimento do *back-end* para o Sistema de Empréstimo de Materiais Acadêmicos da UFERSA cumpriu o objetivo central de prover uma API REST capaz de gerenciar de forma consistente usuários, materiais e o ciclo de empréstimos. Ao longo do projeto, adotou-se uma arquitetura em camadas, com separação clara entre controladores, serviços e repositórios, e empregou-se autenticação baseada em JWT para suporte a operações protegidas. Os resultados apresentados no Capítulo 4 evidenciam que os principais fluxos, cadastro e autenticação de usuários, registro e consulta de materiais, criação, devolução e cancelamento de empréstimos, funcionam conforme esperado, preservando a integridade transacional e entregando respostas padronizadas para sucesso e falha. A documentação interativa via *Swagger*/*OpenAPI*, aliada à coleção Postman, viabiliza a reprodutibilidade das execuções e facilita a integração por terceiros.

Embora o sistema atenda aos requisitos funcionais essenciais e apresente evidências preliminares favoráveis quanto à usabilidade e consistência, algumas restrições metodológicas foram registradas: os testes ocorreram em ambiente local, em parte com configuração de segurança permissiva, e não incluíram cargas elevadas ou cenários de concorrência intensiva. Ainda assim, a ausência de estados inconsistentes nas operações

transacionais e a uniformidade no tratamento de erros reforçam a maturidade da solução em seu estágio atual. Assim, considera-se que o *back-end* se encontra apto para evoluir a fases subsequentes de validação, especialmente em ambientes de staging mais próximos ao contexto de produção.

5.1 Trabalhos futuros

A continuidade do trabalho deve priorizar o endurecimento da camada de segurança e a validação em ambientes mais próximos de produção. Recomenda-se ativar e testar políticas de autorização completas, alinhadas às práticas descritas no OWASP ASVS 4.0 (OWASP, 2023), incluindo verificação de papéis e *ownership* de recursos, além de implementar *rate limiting* e proteções contra força bruta. O ciclo de vida de *tokens* JWT deve ser validado conforme o RFC 7519 (Jones et al., 2015), abrangendo expiração, *refresh* e revogação. Também é pertinente conduzir testes de segurança direcionados (*fuzzing*, análises de *tokens* expirados e inválidos, simulações de ataque) tomando como referência as diretrizes do OWASP *API Security* Top 10 (OWASP, 2023).

Para desempenho e concorrência, propõe-se a realização de testes de carga e estresse com ferramentas como Apache JMeter (*The Apache Software Foundation*, 2023) ou *Locust*, medindo latência, *throughput* e contenção nas operações críticas de empréstimo e devolução. Esses ensaios devem ser repetidos sob diferentes perfis de dados e cenários de concorrência para revelar gargalos transacionais e orientar otimizações no acesso a dados e no uso de bloqueios.

No eixo de observabilidade, a instrumentação com métricas (*Micrometer*) e séries temporais em *Prometheus* (*Prometheus Authors*, 2023) permitirá monitorar tempos de resposta por *endpoint*, taxas de erro e uso de recursos, além de configurar alertas proativos. *Logs* estruturados e eventuais traces distribuídos (caso surja um *front-end* ou serviços adicionais) complementarão a capacidade de diagnóstico e prevenção de regressões.

Quanto à reprodutibilidade e automação, sugere-se consolidar *pipelines* de CI que executem testes automatizados de API e verificações básicas de segurança a cada alteração, bem como *scripts* de *seed* de dados e configuração padronizada de ambiente para facilitar replicações independentes. A documentação das versões de dependências e do JDK, além da fixação de variáveis de ambiente e perfis de execução, reduzirá a variabilidade entre execuções.

Sob o ponto de vista funcional, abrem-se frentes de ampliação do domínio: notificações de prazos de devolução; relatórios gerenciais de uso de materiais; políticas diferenciadas de empréstimo por perfil; suporte a reservas futuras; e integração com sistemas institucionais (SSO/diretórios acadêmicos) para reduzir atrito na autenticação. A disponibilização de um *front-end* ou aplicativo cliente, já apoiado pela especificação OpenAPI existente, pode elevar a adoção pela comunidade acadêmica e fornecer *feedback* adicional sobre usabilidade.

REFERÊNCIAS

ALVES, D. R.; RIBEIRO, R. C. **Gestão de recursos em instituições de ensino. Revista de Administração e Inovação**, v. 13, n. 2, p. 92–104, 2016.

APACHE SOFTWARE FOUNDATION. **Apache JMeter User Manual**. 2023. Disponível em: https://jmeter.apache.org/usermanual/index.html. Acesso em: 28 nov. 2025.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **ABNT. NBR 6023**: informação e documentação - referências - elaboração. Rio de Janeiro:ABNT, 2018.

BECK, K. et al. **Manifesto Ágil: principais valores e princípios**. 2001. Disponível em: https://agilemanifesto.org. Acesso em: 29 nov. 2025.

CHIAVENATO, I. **Administração: teoria, processo e prática**. Rio de Janeiro: Elsevier, 2010.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. 180f. Tese (Doutorado em Ciência da Informação e da Computação) - University of California, Irvine, 2000. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 28 nov. 2025.

GARRETT, J. J. **The elements of user experience: user-centered design for the web and beyond**. Berkeley: New Riders Publishing, 2011.

GOMES, Raquel Silva. **Aplicação do modelo de aceitação da tecnologia (TAM) para analisar os fatores que afetam o uso do Google Classroom entre estudantes do ensino médio**. 2022. 85f. Trabalho de Conclusão de Curso (Graduação em Educação) – Instituto Federal do Espírito Santo, Vitória, 2022. Disponível em: https://repositorio.ifes.edu.br/handle/123456789/3241.

JONES, M. B.; BRADLEY, J.; SAKIMURA, N. JSON Web Token (JWT). RFC 7519. IETF, 2015. Disponível em: https://tools.ietf.org/html/rfc7519. Acesso em: 28 nov. 2025.

MELO, T. R. S.; MORAIS, E. B. D.; DE SOUSA, R. R.; GONÇALVES, S. M. N. Utilizando o método pesquisa-ação para desenvolver um protótipo de um sistema para empréstimos de materiais para a UFERSA, Campus Pau dos Ferros. **Revista de Engenharia e Tecnologia**, v. 17, n. 1, 2025. Disponível em: https://revistas.uepg.br/index.php/ret/article/view/24464.

ORACLE. **O que é um banco de dados?** 2020. Disponível em: https://www.oracle.com/br/database/what-is-database/. Acesso em: 28 nov. 2025.

ORACLE. **The Java Language Specification, Java SE 21 Edition**. 2023. Disponível em: https://docs.oracle.com/javase/specs/. Acesso em: 28 nov. 2025.

OWASP. **OWASP Application Security Verification Standard 4.0 (ASVS)**. 2023. Disponível em: https://owasp.org/www-project-application-security-verification-standard/. Acesso em: 29 nov. 2025.

OWASP. **OWASP API Security Top 10**. 2023. Disponível em: https://owasp.org/www-project-api-security/. Acesso em: 29 nov. 2025.
POSTMAN. **Postman Documentation**. 2024. Disponível em: https://learning.postman.com/docs/. Acesso em: 28 nov. 2025.

PROMETHEUS AUTHORS. **Prometheus Documentation**. 2023. Disponível em: https://prometheus.io/docs/introduction/overview/. Acesso em: 5 dez. 2025.

SILVA, J. A. **Elementos da experiência do usuário by Garrett e a estratégia de validação**. Medium, 2025. Disponível em: https://jeffersonalex.medium.com/elementos-da-experiência-do-usuário-bygarrett-e-a-estratégia-de-validaçãoA3o-96cebc2c6aa5. Acesso em: 5 dez. 2025.