UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE - CCENS
CURSO DE SISTEMAS DE INFORMAÇÃO

KAUA DE SOUZA DA SILVA
MAIK RAMOS MAIFREDO
VINICIUS DE MELLO VIEIRA

RELATÓRIO DO TRABALHO PRÁTICO

COM10616 - SISTEMAS DISTRIBUÍDOS

ALEGRE - ES
2026

KAUA DE SOUZA DA SILVA
MAIK RAMOS MAIFREDO
VINICIUS DE MELLO VIEIRA

RELATÓRIO DO TRABALHO PRÁTICO

COM10616 - SISTEMAS DISTRIBUÍDOS

Trabalho apresentado ao Departamento de Computação, do Centro de Ciências Exatas, Naturais e da Saúde, da Universidade Federal do Espírito Santo, como requisito parcial para a obtenção de aprovação na disciplina de Sistemas Distribuídos.

ALEGRE - ES
2026

Dedico este trabalho a todos que, de alguma forma, contribuíram para meu crescimento pessoal e profissional.

AGRADECIMENTOS

Agradecemos primeiramente à Universidade Federal do Espírito Santo (UFES) e ao Centro de Ciências Exatas, Naturais e da Saúde (CCENS) por proverem a infraestrutura e o ambiente acadêmico necessários para o nosso desenvolvimento.

Ao Departamento de Computação, por manter a excelência no ensino e fomentar o pensamento crítico voltado à inovação tecnológica.

Ao professor da disciplina de Sistemas Distribuídos, por compartilhar seus conhecimentos e orientar este trabalho prático, desafiando-nos a aplicar conceitos teóricos em cenários de desenvolvimento profissional.

Aos nossos colegas de curso, pelas trocas de experiências e auxílio mútuo durante as etapas de depuração e testes. E, finalmente, aos nossos familiares, pelo apoio incondicional que nos permitiu dedicar o tempo necessário à conclusão deste projeto.

"Em algum lugar, algo incrível está esperando para ser conhecido."
Carl Sagan

RESUMO

Este relatório apresenta o desenvolvimento de uma API RESTful para o Sistema de Registros Acadêmicos, requisito da disciplina de Sistemas Distribuídos. A solução foi construída utilizando o framework Quarkus 3.30.8 e Java 21, focando em alta performance e arquitetura cloud-native. O sistema implementa operações CRUD completas para estudantes, disciplinas e registros acadêmicos, integrando persistência em banco de dados MySQL, segurança via JWT (JSON Web Tokens) com criptografia RSA e documentação automatizada com Swagger. Os resultados demonstram uma aplicação robusta, seguindo os princípios da arquitetura em camadas e os requisitos funcionais de paginação, filtragem e autorização RBAC.

Palavras-chave: Quarkus, Java, API REST, Sistemas Distribuídos, Gestão Acadêmica.

LISTA DE ABREVIATURAS E SIGLAS

API - Interface de Programação de Aplicação (Application Programming Interface)
DTO - Objeto de Transferência de Dados (Data Transfer Object)
JWT - Token Web JSON (JSON Web Token)
ORM - Mapeamento Objeto-Relacional (Object-Relational Mapping)
RBAC - Controle de Acesso Baseado em Papéis (Role-Based Access Control)
REST - Transferência de Estado Representacional (Representational State Transfer)
JPA - API de Persistência Java (Java Persistence API)
HTTP - Protocolo de Transferência de Hipertexto (Hypertext Transfer Protocol)
CRUD - Criar, Ler, Atualizar e Excluir (Create, Read, Update, Delete)

SUMÁRIO

1 INTRODUÇÃO

No cenário contemporâneo de Sistemas Distribuídos, a interoperabilidade e a escalabilidade são pilares fundamentais. A transição de arquiteturas monolíticas para microserviços impulsionou o uso de APIs RESTful como o principal meio de comunicação em rede. Este projeto consiste no desenvolvimento de uma API para o Sistema de Registros Acadêmicos, utilizando o framework Quarkus.

O objetivo central foi implementar um sistema que gerencie de forma eficiente e segura os dados de estudantes e disciplinas, garantindo que as operações de rede sigam os princípios de statelessness do REST e fornecendo uma camada de segurança robusta para proteger dados sensíveis. A escolha do Quarkus justifica-se por sua capacidade de compilação nativa e baixo consumo de recursos, características essenciais para sistemas distribuídos modernos que operam em ambientes de nuvem.

2 METODOLOGIA

A metodologia adotada para a construção deste trabalho seguiu as melhores práticas de engenharia de software e os requisitos da disciplina:

- Arquitetura em Camadas: O projeto foi organizado para separar as preocupações de infraestrutura e negócio. A camada api lida com o protocolo HTTP; a core centraliza as regras de negócio e validações; e a data gerencia a persistência via Hibernate ORM.

- Ciclo de Desenvolvimento Ágil: Utilizamos o Git como ferramenta de versionamento, permitindo que os integrantes Kauã, Maik e Vinicius trabalhassem de forma assíncrona em diferentes módulos (Estudantes, Disciplinas e Segurança).

- Metodologia Twelve-Factor App: Implementamos a configuração externa do sistema por meio de variáveis de ambiente (.env), facilitando a portabilidade entre ambientes de desenvolvimento, teste e produção.

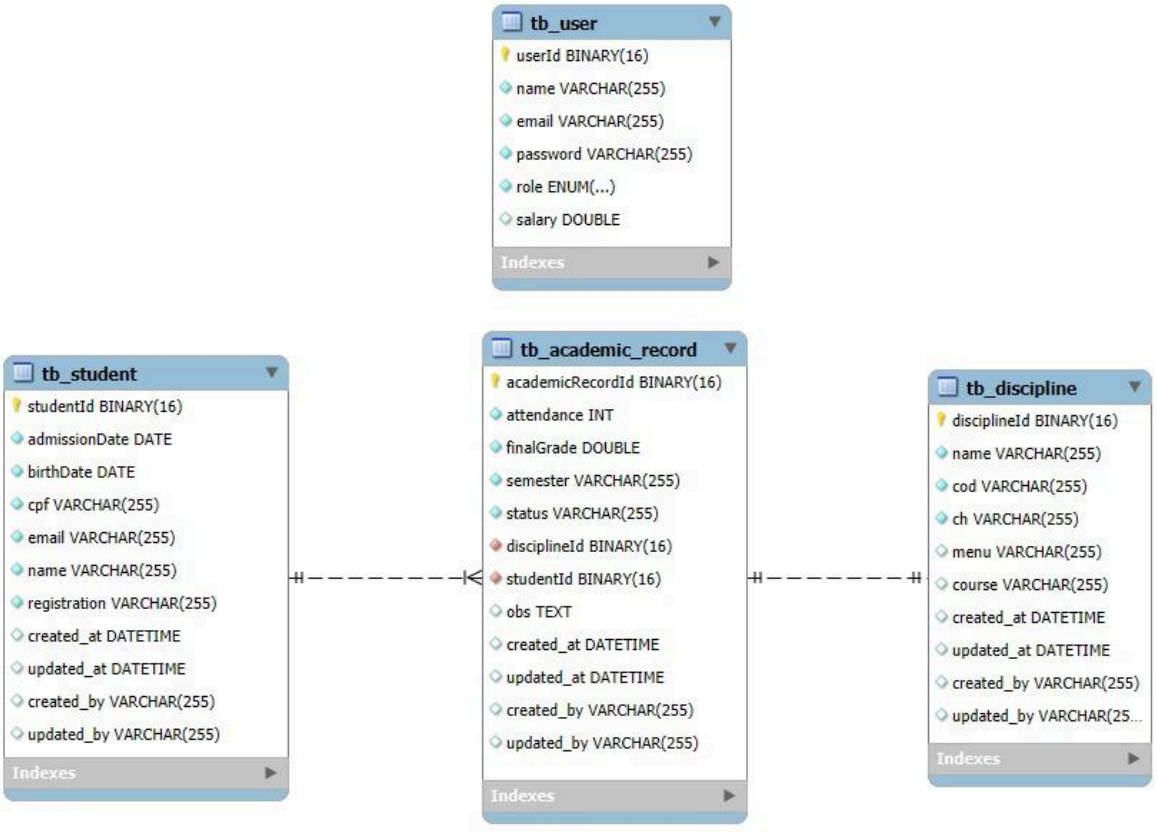
3 MODELAGEM DO SISTEMA

A modelagem do sistema foi concebida para garantir a integridade dos dados acadêmicos e a escalabilidade da API. O projeto utiliza o mapeamento objeto-relacional (ORM) para traduzir a lógica de negócio em uma estrutura de banco de dados SQL robusta.

3.1 DIAGRAMA ENTIDADE-RELACIONAMENTO (ER)

O modelo de dados baseia-se em uma arquitetura que separa a identidade do usuário de suas funções acadêmicas. O relacionamento central ocorre entre Estudantes e Disciplinas, mediado pelo Registro Acadêmico, que funciona como uma entidade associativa de histórico.

Figura 01 - Modelo Entidade-Relacionamento



Fonte: os autores.

As relações principais são:

1. User 1:1 Student: Um estudante é uma especialização de um usuário no sistema.

2. Student 1:N AcademicRecord: Um aluno possui vários registros de desempenho.

3. Discipline 1:N AcademicRecord: Uma disciplina está presente em múltiplos registros de diferentes alunos.

3.2 DETALHAMENTO DAS ENTIDADES E ATRIBUTOS

Conforme os requisitos do trabalho, as entidades possuem atributos obrigatórios para garantir a consistência das informações.

Tabela: Student (Estudante) Responsável por armazenar os dados civis e acadêmicos do discente.

Atributo	Tipo	Restrição	Descrição
id	UUID	PK, Not Null	Identificador único universal.
name	String	Not Null	Nome completo do aluno.
registration	String	Unique, Not Null	Matrícula acadêmica (Ex: 2023100...).
cpf	String	Unique, Not Null	Cadastro de Pessoa Física (validado).
email	String	Unique, Not Null	E-mail institucional ou pessoal.
birthDate	LocalDate	Not Null	Data de nascimento para fins de registro.

admissionDate	LocalDate	Not Null	Data de ingresso no curso.

Tabela: Discipline (Disciplina)

Representa as matérias ofertadas pelo departamento.

Atributo	Tipo	Restrição	Descrição
id	UUID	PK, Not Null	Identificador único.
code	String	Unique, Not Null	Código da disciplina (Ex: COM10616).
name	String	Not Null	Nome da disciplina.
description	String	-	Ementa ou breve descrição.
credits	Integer	Not Null	Carga horária convertida em créditos.
createdAt	LocalDateTime	Not Null	Data de criação do registro no sistema.

Tabela: AcademicRecord (Registro Acadêmico)

Entidade que vincula o desempenho do aluno a uma disciplina.

Atributo	Tipo	Restrição	Descrição
id	UUID	PK, Not Null	Identificador único do registro.
student_id	UUID	FK, Not Null	Referência ao estudante.
discipline_id	UUID	FK, Not Null	Referência à disciplina.
grade	BigDecimal	Not Null	Nota final (intervalo 0.0 a 10.0).

frequency	BigDecimal	Not Null	Porcentagem de presença (0 a 100).
status	Enum	Not Null	Situação (APPROVED, FAILED, IN_PROGRESS).

3.3 PADRÃO ARQUITETURAL E ESTRUTURA DE PASTAS

O sistema adota a Arquitetura em Camadas (Layered Architecture), o que facilita a manutenção e o desacoplamento de componentes, característica vital em sistemas distribuídos.

1. Camada de API (Controller): Gerencia os protocolos de entrada (HTTP) e define os contratos da API. Utiliza DTOs (Data Transfer Objects) para evitar a exposição direta das entidades de banco de dados.

2. Camada Core (Service): Onde reside a inteligência do sistema. Aqui são aplicadas as validações de regra de negócio (Ex: impedir que um aluno seja deletado se possuir registros ativos).

3. Camada de Dados (Repository): Utiliza o padrão Repository com Panache, abstraindo as queries SQL e facilitando o acesso ao banco MySQL.

A estrutura de diretórios reflete essa divisão:
src/main/java/br/ufes/ccens/api: Controllers, Mappers e DTOs.
src/main/java/br/ufes/ccens/core: Services e Regras de Negócio.
src/main/java/br/ufes/ccens/data: Entidades JPA e Interfaces Repository.
src/main/resources: Arquivos de configuração (application.properties) e chaves de segurança.

4 IMPLEMENTAÇÃO E SEGURANÇA

A implementação técnica da API Gestão Acadêmica foi realizada utilizando o ecossistema Quarkus, aproveitando sua alta integração com o Jakarta REST (anteriormente JAX-RS) para a construção de serviços stateless.

4.1 ESPECIFICAÇÃO DOS ENDPOINTS

4.1.1 Gestão de Estudantes (/students)

- Listagem (GET): Implementa paginação e filtragem dinâmica. O sistema utiliza query parameters como page, pageSize, name e cpf.

- Consulta por ID (GET): Recupera os detalhes completos de um aluno específico (RF02).

- Persistência (POST/PUT): Recebe um StudentRequest (DTO), valida os campos via Hibernate Validator e persiste no banco.

4.1.2 Gestão de Disciplinas (/disciplines)

- Controle de Oferta: Gerencia as matérias disponíveis no departamento. A criação (POST) exige um código único (ex: COM10616) e valida a carga horária em créditos.

- Manutenção (PUT/DELETE): Permite a atualização de ementas e nomes, além da remoção de disciplinas que não possuam vínculos ativos.

- Listagem e Filtros: Suporta a busca por nome ou código da disciplina, facilitando a navegação do coordenador acadêmico no sistema distribuído.

4.1.3 Registros Acadêmicos (/academic-records)

- Relacionamentos: Este endpoint permite consultar o histórico por aluno (/students/{id}/records) ou por disciplina (/disciplines/{id}/records).

- Lógica de Negócio: No ato da criação de um registro, o AcademicRecordService valida se a nota está no intervalo [0, 10] e calcula automaticamente o status (Aprovado/Reprovado) com base na nota e frequência, integrando os dados de alunos e disciplinas.

Tabela Resumo de Endpoints:

Método HTTP	Endpoint	Descrição da Funcionalidade	Status de Sucesso
POST	`/auth/login`	Realiza a autenticação do usuário, gerando um *JSON Web Token* (JWT).	`200 OK`
GET	`/students`	Apresenta uma lista paginada dos estudantes cadastrados.	`200 OK`
POST	`/students`	Permite o registro de um novo estudante (Acesso exclusivo para Administradores).	`201 Created`
GET	`/disciplines`	Exibe uma lista paginada das disciplinas disponíveis.	`200 OK`
POST	`/disciplines`	Cadastra uma nova disciplina no sistema (Acesso exclusivo para	`201 Created`

		Administradores).	
GET	/academic-r…	Consulta e visualiza históricos acadêmicos e notas dos estudantes.	200 OK
DELETE	/academic-r…	Remove um registro acadêmico específico, identificado pelo seu {id}.	204 No Cont…

Figura 02 - Exemplo de Resposta (Student)

Fonte: os autores.

4.2 SEGURANÇA E AUTENTICAÇÃO

A segurança foi implementada em duas camadas: Autenticação (quem é o usuário) e Autorização (o que ele pode fazer).

4.2.1 Fluxo de Autenticação JWT:

Utilizamos o padrão SmallRye JWT com criptografia assimétrica RSA.

- Geração: Ao realizar o login, o servidor assina o payload com uma Chave Privada (Private Key) de 2048 bits.

- Validação: A API protege os recursos verificando a assinatura do token recebido no header Authorization: Bearer <token> usando a Chave Pública (Public Key) correspondente.

4.2.2 Controle de Acesso (RBAC):

Utilizamos as anotações @RolesAllowed para restringir operações críticas:

- Role ADMIN: Permissão total (CRUD).

- Role PROFESSOR: Permissão de apenas leitura (GET).

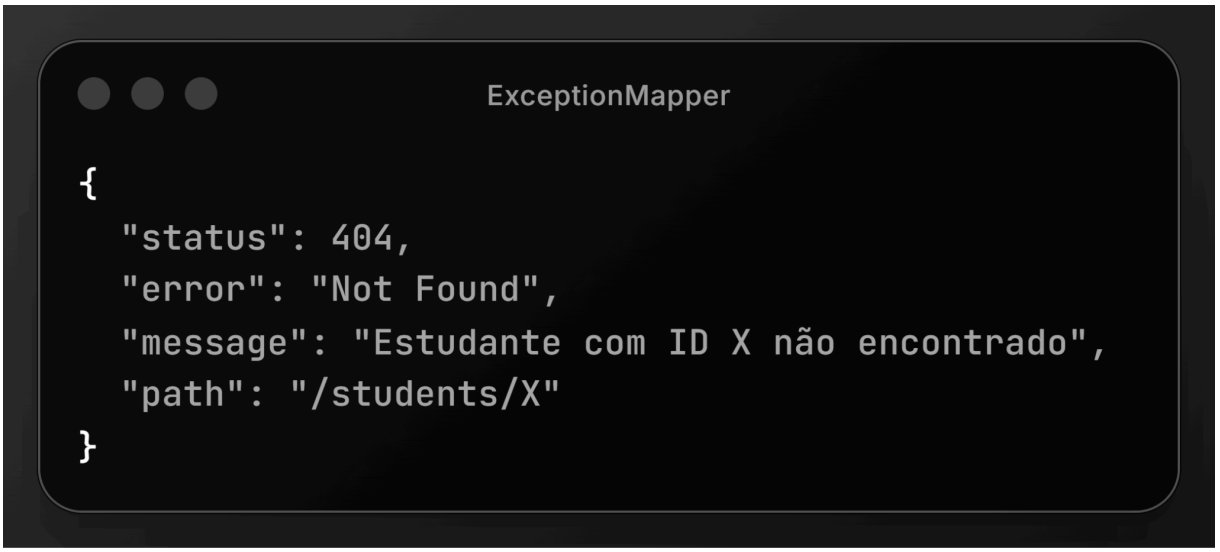
4.2.3 Proteção de Dados:

As senhas dos usuários nunca são armazenadas em texto claro. Utilizamos o algoritmo BCrypt para gerar hashes seguros antes da persistência no MySQL.

4.3 TRATAMENTO DE ERROS E PADRONIZAÇÃO

Para garantir que a API seja amigável ao desenvolvedor (Developer Experience), implementamos um Global Exception Handler utilizando ExceptionMapper. Isso garante que qualquer erro (ex: aluno não encontrado ou erro de banco) retorne um JSON padronizado:

Figura 03 - ExceptionMapper



Fonte: os autores.

4.4 LOGGING E OBSERVABILIDADE

Implementamos a extensão quarkus-logging-json para gerar logs estruturados. Cada transação importante (criação de registros, falhas de login) é capturada por um Interceptor customizado (@LogTransaction), facilitando a auditoria em um ambiente de sistemas distribuídos.

4.5 PLANO DE TESTES

A qualidade do código foi validada através de dois níveis de testes:

- Testes Unitários (JUnit 5 + Mockito): Focados no StudentService para validar regras como a proibição de CPFs duplicados e cálculos de média.
- Testes de Integração (RestAssured): Simulam chamadas HTTP reais para garantir que os endpoints respondam corretamente com os códigos de status REST (200, 201, 400, 401, 404).

5 CONSIDERAÇÕES FINAIS

A conclusão deste trabalho prático permitiu a consolidação de conceitos fundamentais de Sistemas Distribuídos através da implementação de uma API robusta, segura e escalável. O desenvolvimento da API Gestão Acadêmica não se limitou apenas ao cumprimento de requisitos funcionais de CRUD, mas explorou a fundo as camadas de persistência, segurança e arquitetura de software moderna.

5.1 CUMPRIMENTO DOS OBJETIVOS

Todos os objetivos estabelecidos no enunciado foram atingidos com sucesso. A utilização do framework Quarkus 3.30.8 mostrou-se uma escolha estratégica, permitindo que o grupo entregasse uma solução que respeita os princípios de:

- Interoperabilidade: O uso de JSON e padrões RESTful garante que a API possa ser consumida por qualquer cliente (Web, Mobile ou outros microserviços), independentemente da tecnologia utilizada no front-end.

- Segurança Distribuída: A implementação de JWT com chaves RSA assegura que a identidade do usuário seja verificada de forma descentralizada, eliminando a necessidade de manter estados de sessão no servidor (stateless), o que é um pilar de sistemas escaláveis.

5.2 DESAFIOS E LIÇÕES APRENDIDAS

Durante o desenvolvimento, a equipe enfrentou desafios significativos, especialmente na configuração da criptografia assimétrica para o JWT e no mapeamento de entidades complexas com o MapStruct. A necessidade de garantir que as chaves pública e privada fossem geradas e lidas corretamente pelo Quarkus exigiu um entendimento profundo de padrões de segurança e codificação PKCS8. Além disso, a implementação de filtros dinâmicos e paginação exigiu uma integração

refinada entre os Controllers e os Repositórios Panache para manter a performance da aplicação.

5.3 O QUARKUS NO ECOSSISTEMA DE SISTEMAS DISTRIBUÍDOS

A experiência com o Quarkus revelou o potencial do framework no cenário de computação em nuvem. O baixo consumo de memória e o tempo de inicialização quase instantâneo (First Response Time) são diferenciais críticos para sistemas distribuídos que precisam escalar horizontalmente em clusters como o Kubernetes. A facilidade do "Dev Mode" também acelerou drasticamente o ciclo de feedback durante a codificação das regras de negócio acadêmicas.

5.4 TRABALHOS FUTUROS

Como perspectivas de evolução para o projeto, identificamos os seguintes pontos:

- Containerização: Criação de imagens Docker otimizadas para deploy em ambientes orquestrados.

- Esteira de CI/CD: Implementação de testes automatizados integrados ao GitHub Actions para garantir a qualidade em cada commit.

- Monitoramento: Integração com ferramentas como Prometheus e Grafana para observabilidade de métricas de performance e saúde do sistema em tempo real.

- Cachê Distribuído: Implementação de Redis para otimizar as consultas de listagem de estudantes e disciplinas com alta frequência de acesso.

Em suma, o projeto demonstrou que a aplicação de boas práticas de arquitetura e o uso de ferramentas de ponta resultam em um sistema distribuído confiável e pronto para os desafios de um ambiente acadêmico real.

REFERÊNCIAS

HIBERNATE. **Hibernate ORM with Panache**. 2026. Disponível em: https://quarkus.io/guides/hibernate-orm-panache. Acesso em: 17 fev. 2026.

QUARKUS. **Quarkus - Supersonic Subatomic Java**. 2026. Disponível em: https://quarkus.io/. Acesso em: 17 fev. 2026.

RFC 7519. **JSON Web Token (JWT)**. IETF, 2015. Disponível em: https://tools.ietf.org/html/rfc7519. Acesso em: 17 fev. 2026.