Documento de arquitetura de software - Feedback.edu

Projeto: Plataforma para que professores e alunos possam dar feedbacks entre si de forma anônima.
Data: 28/08/2025
Versão: 1.0
Equipe: *Felipe Pereira da Silva*
Samuel Horta de Faria
Rikerson Antonio Freitas Silva

1. Introdução

1. 1 Objetivo

Este documento descreve a arquitetura de software para o sistema feedback.edu. O seu objetivo é servir como um guia técnico para a equipe de desenvolvimento, definindo a estrutura, os componentes principais, as tecnologias e os padrões a serem adotados durante a construção do projeto. A arquitetura aqui definida fornecerá uma base sólida para as decisões técnicas e facilitará a comunicação entre todos os membros da equipe.

1.2 Escopo

O escopo do sistema, para o qual esta arquitetura foi projetada, permitirá a criação de um canal de comunicação entre alunos e professores. As funcionalidades contempladas incluem o cadastro e autenticação de usuários (alunos e professores), a criação e gerenciamento de turmas, o envio de feedbacks anônimos por parte dos alunos e a visualização desses feedbacks pelos professores responsáveis.

2. Visão geral da arquitetura

2.1 Estilo arquitetural

O sistema seguirá o padrão MVC (Model-View-Controller), que é ideal para aplicações web, combinado com uma Arquitetura em Camadas para separar as responsabilidades do back-end. A comunicação entre o front-end e o back-end será feita através de uma API REST. O back-end será desenvolvido em Java utilizando o framework Spring Boot, que agiliza a configuração e o desenvolvimento de serviços RESTful.

2.2 Camadas

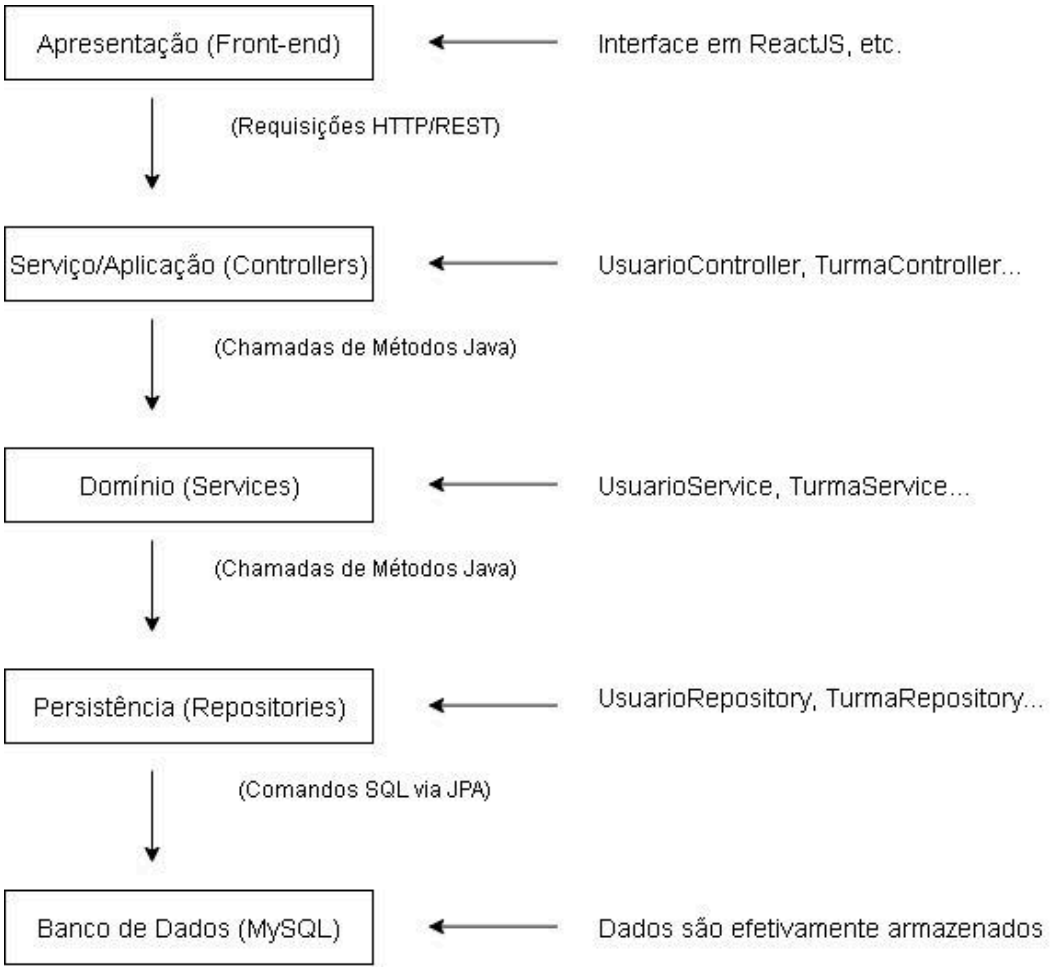
A arquitetura do sistema será dividida nas seguintes camadas, cada uma com sua responsabilidade específica:

- Apresentação (Front-end): Responsável por toda a interface gráfica e interação com o usuário. Será desenvolvida como uma Single-Page Application (SPA).
- Serviço / Aplicação (Controllers): Contém os Controladores REST que servem como a porta de entrada da API. Eles recebem as requisições HTTP do front-end, orquestram as ações e retornam as respostas.
- Domínio (Services): Camada que contém a lógica de negócio principal do sistema, as regras de validação e a manipulação das entidades (como Usuário, Turma e Feedback).
- Persistência (Repositories): Responsável pela comunicação com o banco de dados. Abstrai o acesso aos dados através de Repositórios, utilizando JPA / Hibernate para o mapeamento objeto-relacional.
- Infraestrutura: Camada transversal que contém configurações técnicas, como a conexão com o banco de dados, configurações de segurança (autenticação, etc.) e outros serviços de suporte.

3. Componentes principais

UsuarioController	Expõe os endpoints REST para o cadastro e autenticação dos usuários (alunos e professores).
TurmaController	Gerencia os endpoints REST para a criação de novas turmas (professor) e inscrição de alunos.
FeedbackController	Gerencia os endpoints REST para a submissão de feedbacks (aluno) e a visualização dos mesmos (professor).
UsuarioService	Contém a lógica de negócio para a validação, cadastro e autenticação de usuários.
TurmaService	Contém as regras de negócio para criar turmas, gerar códigos de inscrição únicos e gerenciar a matrícula de alunos.
FeedbackService	Responsável pelas regras de negócio do feedback, incluindo a validação do envio e, crucialmente, a garantia do anonimato ao salvar os dados.
UsuarioRepository	Repositório (JPA) responsável pela persistência e consulta dos dados da entidade *Usuario* no banco de dados.
TurmaRepository	Repositório (JPA) para a persistência e consulta dos dados da entidade *Turma* e das inscrições dos alunos.
FeedbackRepository	Repositório (JPA) para persistência e consulta dos dados da entidade

Diagrama de componentes



4. Tecnologias utilizadas

Camada	Tecnologias
Front-end	ReactJS
Back-end	Java + SpringBoot
Banco de dados	MySQL
ORM	Hibernate / JPA (via Spring Data JPA)
Autenticação	JSON Web Tokens (JWT) para a segurança da API
Versionamento	Git e GitHub

Deploy	Docker para conteinerização da aplicação

5. **Padrões e convenções**

Para garantir a qualidade e a padronização do software, o projeto feedback.edu adotará os seguintes padrões e convenções:

- **Padrão DTO (Data Transfer Object):** Todos os dados trocados entre o front-end e a API REST deverão utilizar objetos DTO. Isso desacopla a representação da API das entidades do banco de dados, aumentando a segurança e a flexibilidade.
- **Padrões Service e Repository:** A lógica de negócio ficará encapsulada na camada de Serviço (Service), enquanto o acesso aos dados será de responsabilidade exclusiva da camada de Repositório (Repository).
- **Isolamento de Camadas por Pacotes:** A estrutura de pacotes do projeto Java deverá refletir a arquitetura em camadas (ex: com.feedbackedu.controller, com.feedbackedu.service, com.feedbackedu.repository, com.feedbackedu.domain).
- **Tratamento de Erros Centralizado:** As exceções e erros da aplicação serão tratados de forma centralizada utilizando a anotação @ControllerAdvice do Spring Boot, provendo respostas de erro padronizadas para a API.
- **Convenção de Nomenclatura para API REST:** Os endpoints da API seguirão os princípios RESTful, utilizando substantivos no plural para os recursos (ex: /turmas, /usuarios) e os verbos HTTP corretos para as ações (GET, POST, PUT, DELETE).
- **Estilo de Código:** O código Java seguirá as diretrizes do Google Java Style Guide para garantir consistência na formatação, nomenclatura e organização do código.

6. **Requisitos não funcionais**

Requisito	**Descrição**
Segurança	As senhas dos usuários serão armazenadas no banco de dados utilizando criptografia (hashing). A comunicação entre o front-end e a API REST será protegida através de autenticação via JWT (JSON Web Tokens).
Manutenibilidade	A Arquitetura em Camadas e a divisão em componentes (Services, Repositories) promovem um código modular, com baixo acoplamento e alta coesão, facilitando futuras manutenções e a adição de novas funcionalidades.
Escalabilidade	A arquitetura desacoplada do back-end, construída com Spring Boot e preparada para conteinerização com Docker, permite a expansão futura

	do sistema. A aplicação pode ser escalada horizontalmente para suportar um aumento no número de usuários.
Anonimato e Privacidade	A arquitetura garante o anonimato ao dissociar a identidade do aluno do seu feedback na Camada de Domínio (FeedbackService) antes de persistir os dados, conforme a regra de negócio principal do sistema.
Disponibilidade	O uso de Docker facilita a implantação da aplicação em ambientes de nuvem robustos (como AWS, Azure, etc.), permitindo configurações de alta disponibilidade para garantir que o sistema permaneça acessível.

7. Riscos arquiteturais

Os principais riscos e desafios técnicos associados à arquitetura proposta são:

- **Gestão da Privacidade e Anonimato:** Sendo o anonimato uma premissa central, uma falha na camada de serviço que, por um bug, permita a associação de um feedback a um aluno comprometeria a confiança e o propósito principal do sistema. A lógica de dissociação deve ser rigorosamente testada.
- **Desempenho sob Carga:** É esperado que o sistema tenha picos de uso, principalmente no final de períodos letivos. Uma alta simultaneidade de envios de feedback pode gerar gargalos na API ou no banco de dados, exigindo futuras otimizações de consultas ou a implementação de uma estratégia de cache.
- **Segurança da API REST:** A exposição de endpoints públicos é um risco inerente. A arquitetura deve ser protegida contra vulnerabilidades comuns, como SQL Injection, Cross-Site Scripting (XSS) e garantir que a implementação do JWT seja robusta para evitar acessos não autorizados.
- **Complexidade do Deploy:** A integração entre um front-end SPA e um back-end conteinerizado com Docker, embora moderna e escalável, pode apresentar desafios na configuração do ambiente de produção e exigir um pipeline de CI/CD (Integração e Entrega Contínua) bem estruturado para garantir implantações suaves.

8. Decisões arquiteturais

Uso de Java com Spring Boot	Framework robusto, amplamente adotado pelo mercado, com uma comunidade sólida e um vasto ecossistema que agiliza o desenvolvimento de APIs REST seguras e eficientes.

Arquitetura em Camadas	Promove uma clara separação de responsabilidades (apresentação, negócio, persistência), resultando em um sistema mais organizado, testável e fácil de manter.
Uso do MySQL como Banco de Dados	Sistema de gerenciamento de banco de dados relacional de código aberto, robusto, confiável e com amplo suporte da comunidade e documentação, adequado para a necessidade do projeto.
Comunicação via API REST	Padrão de mercado para a comunicação entre cliente (front-end) e servidor (back-end). Por ser stateless, facilita a escalabilidade e permite que as duas partes do sistema evoluam de forma independente.
Autenticação via JWT	Padrão moderno e seguro para proteger endpoints em aplicações stateless. Evita a necessidade de gerenciamento de sessão no servidor, o que simplifica a arquitetura e melhora a performance.

9. Diagrama de domínio

Diagrama de Domínio - feedback.edu