

## **LABORATÓRIO 05 - Sistema de Moeda Estudantil (Release 3)**

**Lab05S02**

**Integrantes do Grupo: Guilherme Martini Brina Ferreira e Lucas Nunes Leal Ledsham**

---

### **Análise Crítica**

**Projeto Analisado:** Stuwards

**Repositório GitHub do Projeto:**

[https://github.com/Js3Silva/LDS\\_Moeda\\_Estudantil](https://github.com/Js3Silva/LDS_Moeda_Estudantil)

**Projeto Desenvolvido Por:** Jonathan Sena da Silva, Matheus Fernandes de Oliveira, Victor Gabriel Cruz Pereira e Victor Hugo Dutra Marinho.

---

### **(i) Arquitetura e Tecnologias Utilizadas**

O projeto adota uma arquitetura moderna, bem estruturada e com responsabilidades claramente segregadas. Segue o padrão SPA (Single Page Application) integrado a uma API RESTful no backend.

#### **1) Frontend (React/TypeScript)**

- O uso de TypeScript favorece um desenvolvimento mais seguro e veloz, com acesso a um amplo ecossistema de bibliotecas.
- Tailwind CSS acelera a estilização, embora exija disciplina para evitar excesso de classes no HTML.

- A utilização do Vite (sugerido pela estrutura, apesar do comando npm start, que poderia indicar CRA ou Vite configurado) tende a oferecer desempenho muito superior ao Create-React-App.
- Separação de Responsabilidades: A divisão entre frontend (React) e backend (Spring Boot) está clara e bem definida, permitindo desenvolvimento paralelo, maior desacoplamento e escalabilidade.

## 2) Backend (Java/Spring Boot)

- O uso do Spring Boot é adequado tanto para ambientes acadêmicos quanto corporativos, devido à sua maturidade e robustez.
- A escolha do PostgreSQL é apropriada para dados relacionais estruturados (transações, alunos, professores).
- A inclusão do Lombok reduz código repetitivo e torna o desenvolvimento mais ágil.

### **(ii) Organização do GitHub**

A estrutura do repositório demonstra excelente organização.

- README: Muito bem elaborado, contendo todas as seções essenciais: Visão Geral, Tecnologias, Instalação, Features, Arquitetura e Estrutura de Pastas. O uso de emojis e tabelas torna a leitura mais leve e objetiva.
- Estrutura de Pastas: A separação em project/backend, project/frontend e documents é clara e funcional. Manter artefatos de documentação (UML, requisitos) versionados na pasta documents é uma boa prática.
- Histórico de Commits: Com mais de 170 commits, evidencia desenvolvimento incremental e consistente, evitando grandes entregas únicas.
- Issues: O uso da aba Issues (12 abertas/fechadas) demonstra algum nível de gestão de tarefas e rastreamento de bugs.

### **(iii) Dificuldade para Configuração do Ambiente**

A configuração do ambiente não é difícil, mas também não é totalmente simples.

#### 1) Pontos Fortes

- Docker no Backend: O uso de docker-compose para subir o banco de dados e o backend simultaneamente elimina a necessidade de instalações manuais ou configurações complexas.
- Pré-requisitos Claros: A lista de requisitos (Java 17, Node, Docker) está devidamente especificada.

#### 2) Pontos Fracos

- Configuração Híbrida: O backend roda em container, mas o frontend depende de execução local via npm start, exigindo que o desenvolvedor tenha Node.js instalado na versão adequada. Isso aumenta o esforço de configuração. Se o frontend também fosse containerizado ou hospedado em nuvem (ex.: Azure), a experiência seria mais fluida.
- Variáveis de Ambiente: O README menciona um arquivo .env, mas não detalha seu conteúdo nem indica a existência de um .env.example. Se o docker-compose utilizar variáveis não versionadas, o projeto pode não funcionar na primeira execução.

### **(iv) Sugestões de Melhorias**

As principais recomendações de aprimoramento são:

#### 1) Dockerização Completa

Criar um Dockerfile para o frontend e integrá-lo ao docker-compose.yml. Assim, todo o sistema (Frontend + Backend + Banco) poderia ser iniciado com um único comando (docker-compose up).

Alternativamente, hospedar o frontend em serviços como Azure Static Web Apps também resolveria a dependência do Node local.

## 2) Testes Automatizados (QA)

Atualmente, o projeto menciona testes manuais via Postman. Seria ideal implementar:

- JUnit no backend,
- Jest/Vitest no frontend,

garantindo que regras de negócio críticas (ex.: impedir transferência com saldo insuficiente) permaneçam válidas em futuras atualizações.

## 3) Validação de Dados

Campos sensíveis, como CPF e e-mail, não passam por validação. Adicionar verificações adequadas aumentaria a confiabilidade e integridade dos dados.