Sistema de Moeda Estudantil

Laboratório de Desenvolvimento de Software - Sprint 3

Henrique Lima, João Vitor Ferreira, Pedro Henrique Novais

Visão Geral do Sistema

Objetivo: Sistema de reconhecimento estudantil através de moedas virtuais distribuídas por professores e trocadas por vantagens oferecidas por empresas parceiras.

Usuários do Sistema:

- Alunos - Recebem moedas dos professores e resgatam vantagens
- Professores - Distribuem moedas aos alunos e recebem crédito semestral
- Empresas Parceiras - Cadastram e gerenciam vantagens
- Instituições de Ensino - Vincula alunos e professores
- Administradores - Gerenciam professores e atribuem moedas semestrais

Tecnologias Utilizadas:

- Linguagem & Framework: Java 21 · Spring Boot 3.5.5
- Dados: Spring Data JPA · MySQL
- Segurança: Spring Security (JWT) · BCrypt
- Build & Testes: Maven · JUnit · Spring Security Test
- Documentação: SpringDoc OpenAPI (Swagger)
- Utilitários: Lombok · Bean Validation
- Agendamento: Spring Scheduling (Jobs)

Funcionalidades Principais:

- Autenticação e autorização com JWT
- Cadastro público de alunos e empresas
- Pré-cadastro administrativo de professores
- Gerenciamento de instituições de ensino
- Distribuição de moedas por professores
- Resgate de vantagens por alunos
- Atribuição automática de moedas semestrais (1.000 moedas)
- Consulta de extratos (alunos e professores)
- Gerenciamento de catálogo de vantagens
- Sistema de transações completo

Arquitetura MVC

1. CLIENT (Frontend)
- Interface HTML/JavaScript
- Comunicação via REST API
- Autenticação JWT

2. CONTROLLER (REST API)
- 8 Controllers implementados: `AuthController` - `AdminController` - `AlunoController` - `ProfessorController` - `EmpresaParceiraController` - `InstituicaoEnsinoController` - `TransacaoController` - `VantagemController`

3. SERVICE (Business Logic)
- 8 Services com @Transactional: `AlunoService` - `ProfessorService` - `EmpresaParceiraService` - `InstituicaoEnsinoService` - `TransacaoService` - `VantagemService` - `SemestreService` - `MoedaSemestralService`

4. REPOSITORY (Data Access)
- 9 Repositories extends JpaRepository: `UsuarioRepository` - `AlunoRepository` - `ProfessorRepository` - `EmpresaParceiraRepository` - `InstituicaoEnsinoRepository` - `TransacaoRepository` - `VantagemRepository` - `SemestreRepository` - `AtribuicaoMoedaRepository`

5. MODEL (Database Entities)
- 9 Entidades JPA com relacionamentos: `Usuario` - `Aluno`, `Professor`, `EmpresaParceira` (herdam de Usuario) - `InstituicaoEnsino` - `Transacao` - `Vantagem` - `Semestre` - `AtribuicaoMoeda`

Modelo de Dados - Entidades JPA

Hierarquia de Usuários (Herança JOINED)

Usuario (Classe Abstrata)

- `id: Long`
- `nome: String` (obrigatório)
- `email: String` (obrigatório, único)
- `senha: String` (obrigatório, criptografada com BCrypt)

Aluno (extends Usuario)

- `cpf: String` (obrigatório, único)
- `rg: String` (obrigatório)
- `endereco: String` (obrigatório)
- `curso: String` (obrigatório)
- `saldo: Double` (padrão: 0.0)
- `@ManyToOne InstituicaoEnsino instituicao` (obrigatório)

Professor (extends Usuario)

- `cpf: String` (obrigatório, único)
- `departamento: String` (obrigatório)
- `saldo: Double` (padrão: 0.0)
- `@ManyToOne InstituicaoEnsino instituicao` (obrigatório)

EmpresaParceira (extends Usuario)

- `cnpj: String` (obrigatório, único)
- `@OneToMany List<Vantagem> vantagens`

Modelo de Dados - Entidades JPA

Entidades Independentes

InstituicaoEnsino

- `id: Long`
- `nome: String` (obrigatório)
- `cnpj: String` (obrigatório, único)
- `endereco: String` (obrigatório)
- `telefone: String`
- `email: String`
- `@OneToMany List<Aluno> alunos`
- `@OneToMany List<Professor> professores`

Transacao

- `id: Long`
- `quantidade: Double` (obrigatório, positivo)
- `motivo: String` (obrigatório, max 500 chars)
- `data: LocalDateTime` (gerada automaticamente)
- `tipo: String` (ENVIO, RECEBIMENTO, RESGATE)
- `@ManyToOne Professor professor` (opcional)
- `@ManyToOne Aluno aluno` (opcional)
- `@ManyToOne Vantagem vantagem` (opcional)

Modelo de Dados - Entidades JPA

Entidades Independentes

Vantagem

- `id: Long`
- `descricao: String` (obrigatório, max 500 chars)
- `custo: Double` (obrigatório, positivo)
- `fotoUrl: String`
- `codigoCupom: String` (único, gerado automaticamente no resgate)
- `@ManyToOne EmpresaParceira empresa` (obrigatório)

AtribuicaoMoeda

- `id: Long`
- `@ManyToOne Professor professor` (obrigatório)
- `@ManyToOne Semestre semestre` (obrigatório)
- `quantidade: Double` (padrão: 1000.0)
- `dataAtribuicao: LocalDateTime` (gerada automaticamente)

Semestre

- `id: Long`
- `ano: String` (obrigatório, 4 chars)
- `periodo: String` (obrigatório, "1" ou "2")
- `dataInicio: LocalDateTime` (obrigatório)
- `dataFim: LocalDateTime`
- `ativo: Boolean` (padrão: true)
- `dataCriacao: LocalDateTime` (gerada automaticamente)

Relacionamentos Entre Entidades

Entidade	Relacionamento	Descrição
Aluno	@ManyToOne	InstituicaoEnsino (N alunos : 1 instituição)
Professor	@ManyToOne	InstituicaoEnsino (N professores : 1 instituição)
InstituicaoEnsino	@OneToMany	Alunos e Professores (1 : N)
Vantagem	@ManyToOne	EmpresaParceira (N vantagens : 1 empresa)
EmpresaParceira	@OneToMany	Vantagens (1 empresa : N vantagens)
Transacao	@ManyToOne	Professor, Aluno, Vantagem
AtribuicaoMoeda	@ManyToOne	Professor e Semestre

Controllers REST - Endpoints Implementados

Método HTTP	Endpoint	Descrição
GET	/api/{recurso}	Listar todos
GET	/api/{recurso}/{id}	Buscar por ID
POST	/api/{recurso}	Criar novo
PUT	/api/{recurso}/{id}	Atualizar existente
DELETE	/api/{recurso}/{id}	Remover

Recursos Disponíveis:

/api/alunos • /api/professores • /api/empresas • /api/instituicoes • /api/transacoes • /api/vantagens

Fluxo Completo de uma Requisição

Exemplo 1: POST /auth/login

- Cliente ➜ Envia JSON com `{email, senha}`
- AuthController.login() ➜ Recebe `@RequestBody LoginRequest`
- UsuarioRepository ➜ Busca usuário por email
- PasswordEncoder ➜ Valida senha com BCrypt
- TokenService ➜ Gera JWT token
- AuthController ➜ Retorna `AuthResponse` com token, id, tipo, nome, email
- Cliente ➜ Armazena token em localStorage

OBRIGADO!