Documentação de Projeto

para o sistema

Sistema-de-Moeda-Estudantil

Versão 1.0

Projeto de sistema elaborado pelo(s) aluno(s) Gabriel Burdignon, Vinicius Mazzoli, Matheus Santos
como parte da disciplina **Projeto de Software**.

16/11/2025

Tabela de Conteúdo

Histórico de Revisões

Nome	Data	Razões para Mudança	Versão

1. Introdução

Este documento apresenta os modelos de requisitos, projeto e dados do Sistema de Moeda Estudantil, uma aplicação web criada para estimular o reconhecimento de mérito estudantil por meio de uma moeda virtual.

A proposta do sistema é permitir que professores das instituições parceiras distribuam moedas a seus alunos, como forma de reconhecimento por bom desempenho acadêmico, participação, comportamento e demais critérios definidos pela instituição. Os alunos, por sua vez, podem trocar essas moedas por produtos e descontos oferecidos por empresas parceiras.

Principais objetivos do sistema:

- Incentivar o engajamento dos alunos nas atividades acadêmicas.
-
- Criar um mecanismo transparente de reconhecimento de mérito.
-
- Estabelecer um ecossistema entre instituições, professores, alunos e empresas parceiras.
-
- Registrar e disponibilizar o histórico de transações de moedas para todos os perfis de usuário relevantes.

A implementação do sistema utiliza uma arquitetura web moderna, com frontend em React/Next.js, backend em Node.js e persistência de dados em banco relacional (ex.: PostgreSQL) com ORM (Prisma), oferecendo uma base estruturada para evolução do produto.

2. Modelos de Usuário e Requisitos

2.1 Descrição de Atores

Aluno

- Deseja participar do sistema de mérito estudantil.
-
- Recebe moedas dos professores.
-
- Consulta saldo e extrato de transações.
-
- Troca moedas por vantagens (produtos/descontos) em empresas parceiras.
-

- Recebe notificações por e-mail sobre cadastros, recebimento de moedas e cupons de vantagens.

Professor

- É previamente cadastrado pela instituição.
-
- Possui um "saldo de moedas" por semestre para distribuir aos alunos.
-
- Envia moedas a alunos, informando valor e motivo.
-
- Consulta o próprio extrato de transações (envios realizados).

Instituição de Ensino

- Cadastra e gerencia os professores vinculados.
-
- Garante que os professores estejam corretamente associados à instituição e a um departamento.
-
- Pode consultar relatórios agregados de uso das moedas (opcional / futuro).

Empresa Parceira

- Cadastra sua empresa no sistema.
-
- Informa dados como nome, CNPJ, contato e e-mail.
-
- Cadastra, edita, remove ou desativa vantagens (produtos/descontos).
-
- Recebe cupons gerados quando um aluno troca moedas.
-
- Pode validar o código do cupom (processo de validação é previsto conceitualmente).

Administrador do Sistema

Gerencia o cadastro e os perfis de acesso.

Garante que apenas usuários autenticados acessem suas áreas.

Tem visão global do sistema (instituições, usuários, empresas e transações).

Sistema de E-mail / Serviço Externo de Notificações

Envia e-mails automatizados para confirmação de cadastro, recebimento de moedas e emissão de cupons.

Atua como ator externo, chamado pelo backend do sistema.

2.2 Modelo de Casos de Uso

Nesta subseção são descritos os principais casos de uso, alinhados às **Histórias de Usuário da Sprint 1**, conforme documentado no README do repositório do projeto.

UC-01 / US01 – Cadastrar Aluno
Como aluno, desejo realizar meu cadastro informando **nome, e-mail, CPF, RG, endereço, instituição e curso** para participar do sistema e receber moedas.
Principais regras:

- A instituição deve estar previamente cadastrada.
- Todos os campos obrigatórios devem ser preenchidos.
- Um e-mail de confirmação é enviado ao aluno após o cadastro.

UC-02 / US02 – Cadastrar Empresa Parceira
Como empresa parceira, desejo cadastrar minha empresa informando **nome, CNPJ, e-mail e vantagens oferecidas**, para disponibilizar produtos ou descontos.
Regras:

- O CNPJ deve ser validado.
- A empresa pode cadastrar múltiplas vantagens.
- Cada vantagem contém **descrição, custo em moedas e imagem/foto**.

UC-03 / US03 – Cadastrar Professor pela Instituição
Como instituição, desejo cadastrar previamente os professores com **nome, CPF e departamento**, vinculando-os à instituição.
Regras:

- Todo professor deve estar associado a uma instituição.

- Cada professor deve possuir **login e senha** para acesso ao sistema.

UC-04 / US04 – Autenticar Usuário (Login)
Como usuário (aluno, professor ou empresa), desejo fazer login com meu **e-mail e senha**.
Regras:

- O sistema valida as credenciais.
- Em caso de erro, exibe mensagem adequada.
- A sessão expira após período de inatividade.

UC-05 / US05 – Enviar Moedas a Aluno
Como professor, desejo enviar moedas a um aluno informando **valor e motivo**, para recompensar seu desempenho.
Regras:

- O professor só pode enviar moedas se tiver saldo suficiente.
- O aluno recebe notificação por e-mail.
- A transação é registrada no extrato de ambos.

UC-06 / US06 – Consultar Extrato de Moedas
Como aluno ou professor, desejo visualizar meu **extrato de moedas**, para acompanhar saldo e histórico de transações.
Regras:

- O extrato deve exibir todas as transações (envios, recebimentos, trocas).
- O saldo atualizado precisa estar visível de forma destacada.

UC-07 / US07 – Resgatar Vantagem
Como aluno, desejo **trocar minhas moedas por produtos ou descontos** oferecidos por empresas parceiras.
Regras:

- O aluno deve ter saldo suficiente.
- Após a troca, o saldo é atualizado.

- Um **cupom com código** é enviado por e-mail ao aluno e à empresa.

UC-08 / US08 – Gerenciar Vantagens da Empresa
Como empresa parceira, desejo **cadastrar, editar, excluir ou desativar vantagens** oferecidas.
Regras:

- Toda vantagem possui **nome, descrição, valor em moedas e imagem**.
- A empresa pode desativar temporariamente uma vantagem para que não apareça aos alunos.

UC-09 / US09 – Enviar Notificações Automáticas por E-mail
Como sistema, desejo enviar notificações por e-mail para confirmar cadastros, informar recebimento de moedas e envio de cupons.
Regras:

- E-mails devem conter informações claras, incluindo código de verificação/cupom.
- Devem ser disparados em tempo quase real após cada evento.

UC-10 / US10 – Garantir Acesso Seguro
Como administrador, desejo garantir que somente usuários autenticados acessem suas áreas, protegendo os dados do sistema.
Regras:

- Perfis de acesso diferenciados (aluno, professor, empresa, instituição, admin).
- Acesso às funcionalidades condicionado ao tipo de usuário.

2.3 Diagrama de Sequência do Sistema

Contrato 1 – Operação `cadastrarAluno(dadosAluno)`

- **Operação:** cadastrarAluno(dadosAluno)
- **Atores envolvidos:** Aluno, Sistema, Serviço de E-mail
- **Referências cruzadas:** UC-01 / US01
- **Pré-condições:**

- Instituição selecionada está cadastrada no sistema.
- Campos obrigatórios preenchidos e CPF/e-mail válidos.

- **Pós-condições:**
  - Um registro de aluno é criado no banco de dados com status “ativo/pendente de confirmação” (conforme implementação).
  - Um e-mail de confirmação é enviado ao aluno.
  - O aluno passa a poder efetuar login após eventual confirmação, caso exista fluxo de verificação.

Contrato 2 – Operação `cadastrarEmpresa(dadosEmpresa)`

- **Operação:** cadastrarEmpresa(dadosEmpresa)
- **Atores:** Empresa Parceira, Sistema
- **Referências cruzadas:** UC-02 / US02
- **Pré-condições:**
  - CNPJ informado em formato válido.
- **Pós-condições:**
  - Empresa é persistida no banco de dados com status “ativa”.
  - Usuário responsável pela empresa passa a poder acessar sua área e gerenciar vantagens.

Contrato 3 – Operação `autenticarUsuario(email, senha)`

- **Operação:** autenticarUsuario(email, senha)
- **Atores:** Aluno, Professor, Empresa, Instituição, Administrador
- **Referências cruzadas:** UC-04 / US04, UC-10 / US10

- **Pré-condições:**
    - O usuário já foi previamente cadastrado.
- **Pós-condições:**
    - Em caso de sucesso, é criada uma sessão autenticada com o perfil adequado.
    - Em caso de falha, é retornada mensagem de erro sem criar sessão.

Contrato 4 – Operação `enviarMoedas(professorId, alunoId, valor, motivo)`

- **Operação:** enviarMoedas(professorId, alunoId, valor, motivo)
- **Atores:** Professor, Aluno, Sistema, Serviço de E-mail
- **Referências cruzadas:** UC-05 / US05
- **Pré-condições:**
    - Professor autenticado.
    - Professor possui saldo de moedas suficiente.
    - Aluno existe e está ativo.
- **Pós-condições:**
    - Saldo do professor é decrementado pelo valor enviado.
    - Saldo do aluno é incrementado.
    - Uma transação é registrada no histórico dos dois usuários.
    - Um e-mail é enviado ao aluno notificando o recebimento.

Contrato 5 – Operação `consultarExtrato(usuarioId, filtroOpcional)`

- **Operação:** consultarExtrato(usuarioId, filtroOpcional)
- **Atores:** Aluno ou Professor

- **Referências cruzadas:** UC-06 / US06
- **Pré-condições:**
    - Usuário autenticado.
- **Pós-condições:**
    - O sistema retorna a lista de transações (envios, recebimentos, trocas) associadas ao usuário.
    - Retorna o saldo atual calculado.

Contrato 6 – Operação `resgatarVantagem(alunoId, vantagemId)`

- **Operação:** resgatarVantagem(alunoId, vantagemId)
- **Atores:** Aluno, Empresa Parceira, Sistema, Serviço de E-mail
- **Referências cruzadas:** UC-07 / US07
- **Pré-condições:**
    - Aluno autenticado.
    - Vantagem está ativa.
    - Aluno possui saldo em moedas ≥ custo da vantagem.
- **Pós-condições:**
    - O saldo do aluno é decrementado pelo custo da vantagem.
    - É criado um registro de cupom associado ao aluno, à empresa e à vantagem.
    - E-mails com o código do cupom são enviados ao aluno e à empresa.

Contrato 7 – Operação `gerenciarVantagem(empresaId, dadosVantagem, ação)`

- **Operação:**

- criarVantagem, editarVantagem, excluirVantagem, alterarStatusVantagem

- **Atores:** Empresa Parceira
- **Referências cruzadas:** UC-08 / US08
- **Pré-condições:**
  - Empresa autenticada.
- **Pós-condições:**
  - O catálogo de vantagens da empresa é atualizado conforme a ação realizada.

Formato para cada contrato de operação

Contrato	
Operação	
Referências cruzadas	
Pré-condições	
Pós-condições	

3. Modelos de Projeto

3.1 Arquitetura

A arquitetura do Sistema de Moeda Estudantil segue um modelo **em camadas** com separação entre apresentação, lógica de negócio e acesso a dados:

- **Camada de Apresentação (Frontend):**
  - Implementada em **React/Next.js**.
  - Responsável pelas telas de cadastro, login, envio de moedas, extrato, catálogo de vantagens, painel da empresa, etc.
  - Faz chamadas às APIs REST/HTTP do backend.
- **Camada de Aplicação / Negócio (Backend):**

    - Implementada em **Node.js**, integrada ao Next.js (API routes) ou em servidor separado.

    - Exponibiliza endpoints para gerenciamento de usuários, autenticação, envio de moedas, extratos e resgates de vantagens.

    - Implementa regras de negócio: verificação de saldo, validação de acesso por perfil, geração de cupons, etc.

- **Camada de Persistência (Banco de Dados):**

    - Usa **banco relacional** (por exemplo, PostgreSQL).

    - Acesso via ORM (como Prisma), com mapeamento das entidades de domínio para tabelas.

    - Garante integridade referencial entre usuários, instituições, empresas, vantagens, transações e cupons.

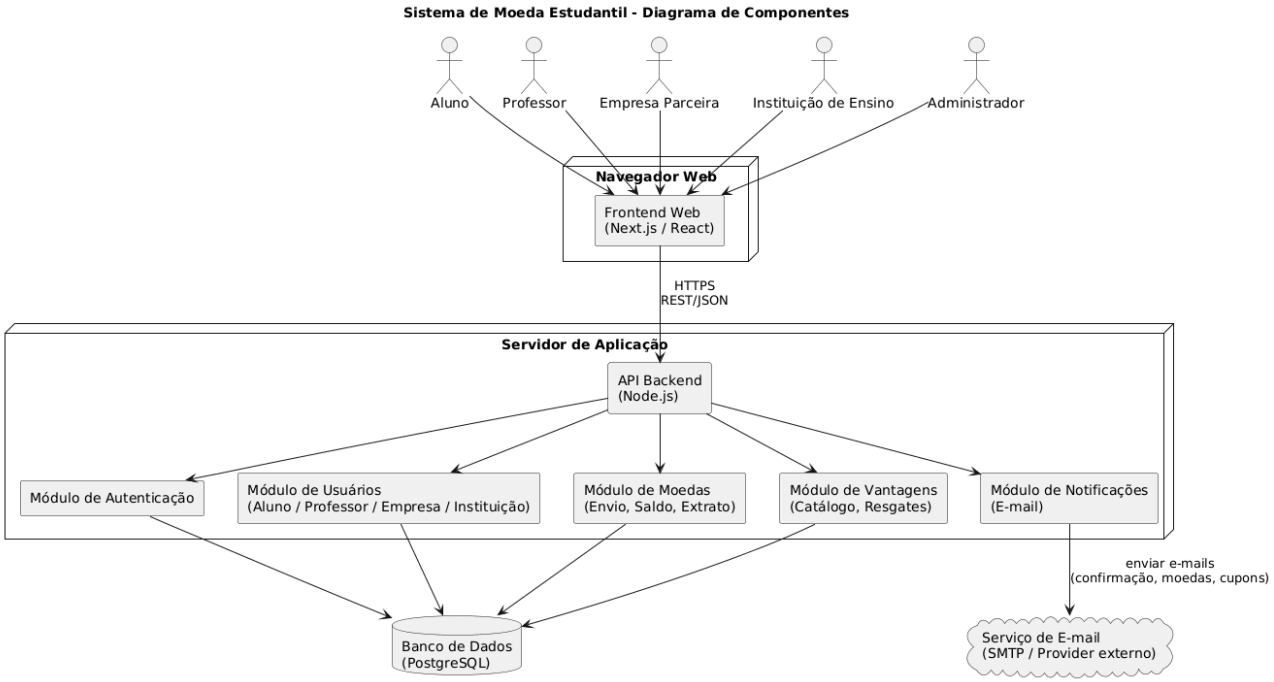
- **Integrações Externas:**

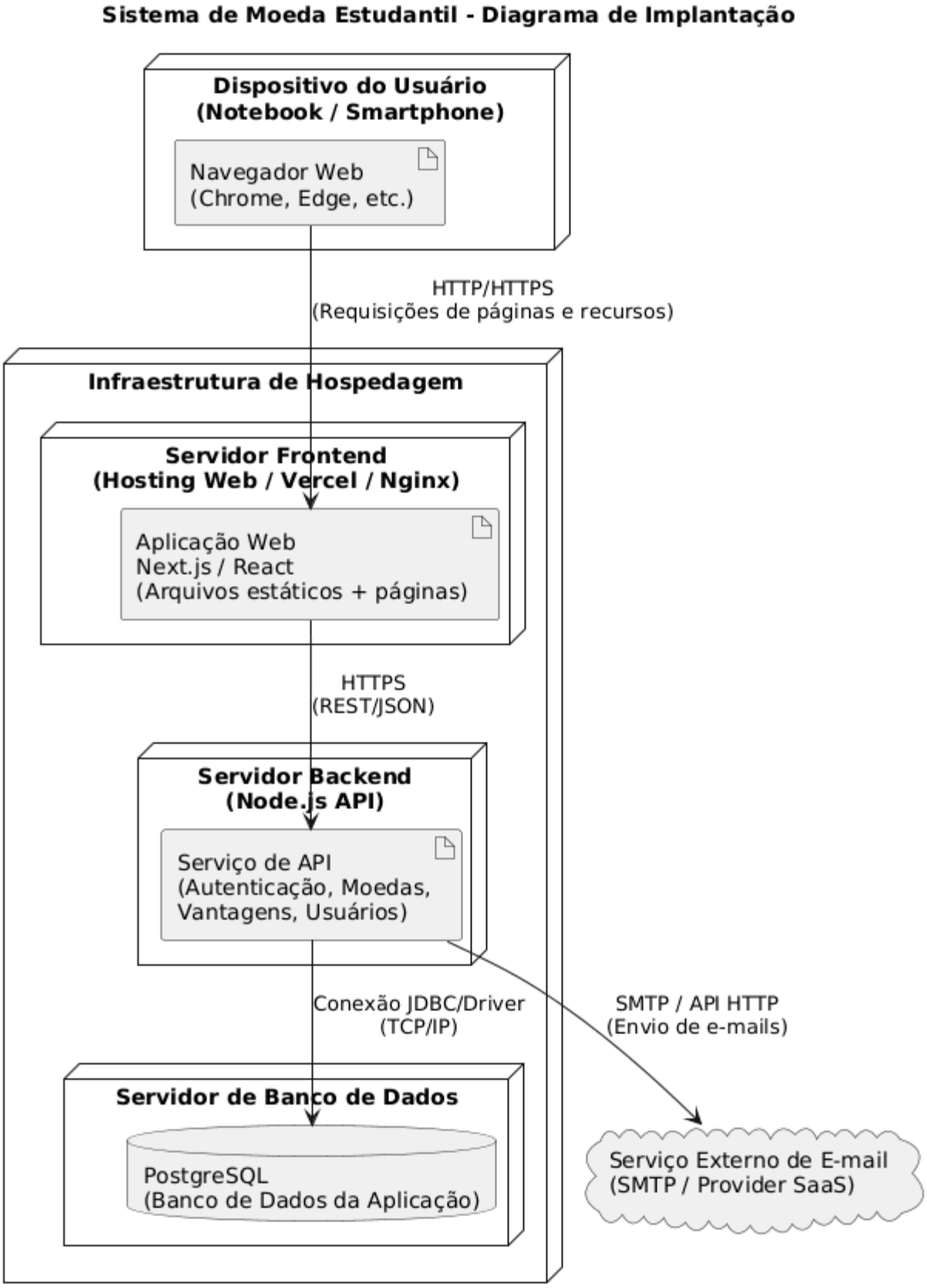
    - Serviço de envio de e-mails (SMTP / serviço externo).

    - Possível integração futura com sistemas da instituição ou das empresas parceiras para validação de matrículas, etc.

Essa separação facilita **evolução, manutenção e testes**, permitindo alterar, por exemplo, o serviço de e-mail ou o banco de dados com impacto reduzido nas demais camadas.

3.2 Diagrama de Componentes e Implantação.




Sistema de Moeda Estudantil - Diagrama de Implantação
Dispositivo do Usuário
(Notebook / Smartphone)
Navegador Web
(Chrome, Edge, etc.)
HTTP/HTTPS
(Requisições de páginas e recursos)
Infraestrutura de Hospedagem
Servidor Frontend
(Hosting Web / Vercel / Nginx)
Aplicação Web
Next.js / React
(Arquivos estáticos + páginas)
HTTPS
(REST/JSON)
Servidor Backend
(Node.js API)
Serviço de API
(Autenticação, Moedas,
Vantagens, Usuários)
Conexão JDBC/Driver
(TCP/IP)
SMTP / API HTTP
(Envio de e-mails)
Servidor de Banco de Dados
PostgreSQL
(Banco de Dados da Aplicação)
Serviço Externo de E-mail
(SMTP / Provider SaaS)

Componente Web Client (Frontend):

- Responsável pelas interfaces de aluno, professor, instituição, empresa e administrador.
- Executa no navegador do usuário.

Componente API / Backend:

- Autenticação e autorização.
- Operações de cadastro e atualização de dados.
- Regras de negócio de moedas e vantagens.
- Comunicação com o banco de dados e serviço de e-mails.

Componente Banco de Dados:

- Contém todas as tabelas relacionadas a usuários, instituições, empresas, vantagens, transações e cupons.

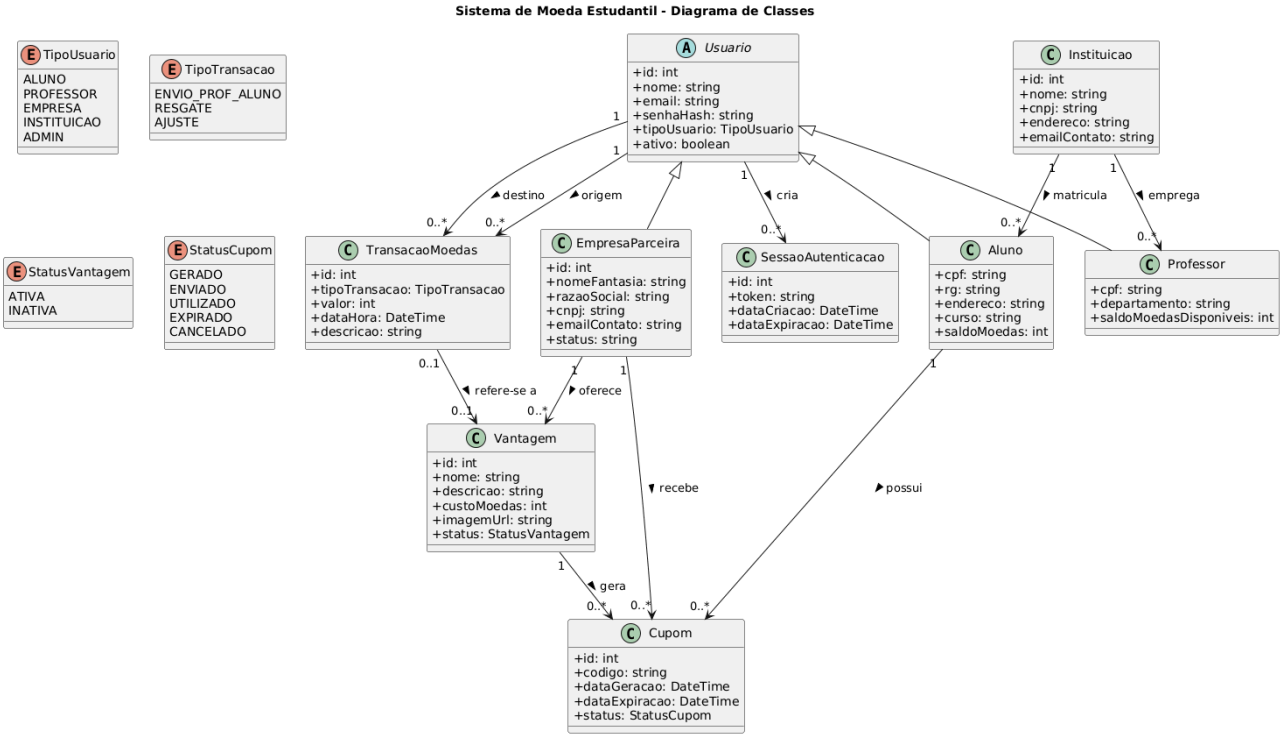
Componente de Notificações (E-mail):

- Serviço SMTP ou API de terceiros (ex.: SendGrid, Mailgun, etc.) consumido pelo backend.

Implantação (visão textual):

- O **Frontend** é implantado em um provedor de hospedagem de aplicações web (ex.: Vercel, Netlify ou servidor HTTP).
- O **Backend** é implantado em um servidor Node.js (podendo ou não ser integrado ao mesmo ambiente do frontend).
- O **Banco de Dados** é executado em um servidor gerenciado ou local, acessível somente pelo backend.
- O **Serviço de E-mail** é externo ao sistema, acessado via internet.

3.3 Diagrama de Classes



A seguir são descritas as principais classes/entidades de domínio:

Usuário (abstrata)

- Atributos: id, nome, email, senhaHash, tipoUsuario (ALUNO, PROFESSOR, EMPRESA, INSTITUICAO, ADMIN), ativo.

Aluno (especializa Usuário)

- Atributos adicionais: cpf, rg, endereco, curso, instituicaoId, saldoMoedas.

Professor (especializa Usuário)

- Atributos adicionais: cpf, departamento, instituicaoId, saldoMoedasDisponiveis.

Instituicao

- Atributos: id, nome, cnpjOpcional, endereco, emailContato.

- Relacionamentos: 1:N com Professor, 1:N com Aluno.

EmpresaParceira (especializa Usuário ou associada à tabela própria)

- Atributos: id, nomeFantasia, razaoSocial, cnpj, emailContato, status.
- Relacionamentos: 1:N com Vantagem (benefícios oferecidos).

Vantagem

- Atributos: id, empresaId, nome, descricao, custoMoedas, imagemUrl, status (ATIVA/INATIVA).

TransacaoMoedas

- Atributos: id, tipoTransacao (ENVIO_PROF_ALUNO, RESGATE, AJUSTE, etc.), valor, dataHora, idUsuarioOrigem, idUsuarioDestino, descricao.
- Relacionamentos: vincula alunos, professores e, indiretamente, vantagens (quando se trata de resgate).

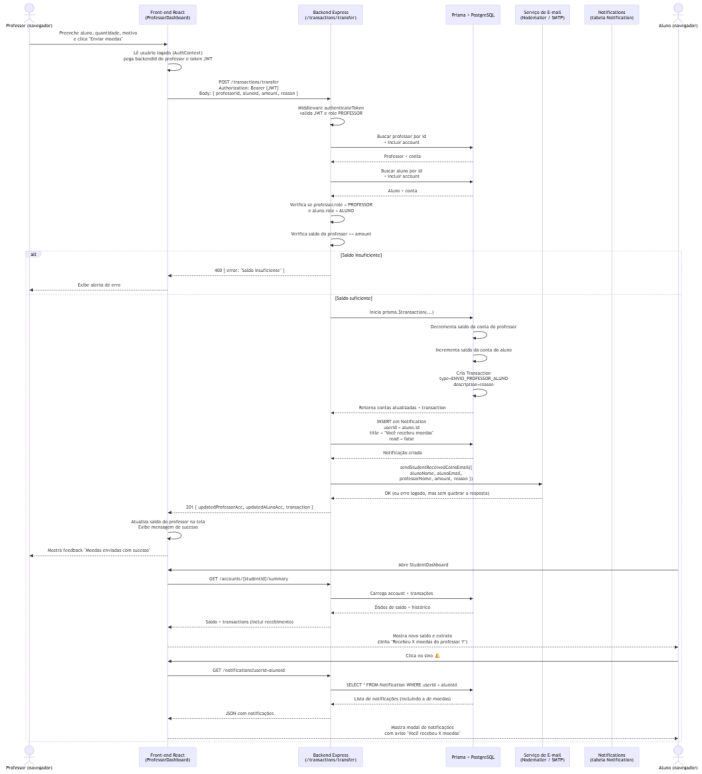
Cupom

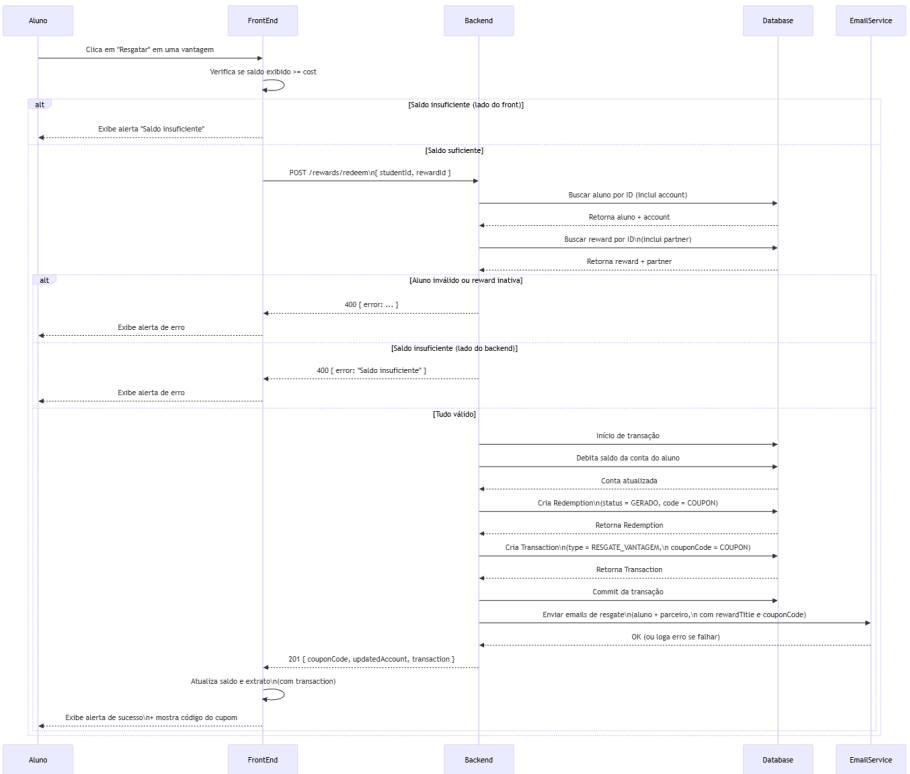
- Atributos: id, codigo, alunoId, empresaId, vantagemId, dataGeracao, dataExpiracao, status (GERADO, ENVIADO, UTILIZADO, EXPIRADO, CANCELADO).

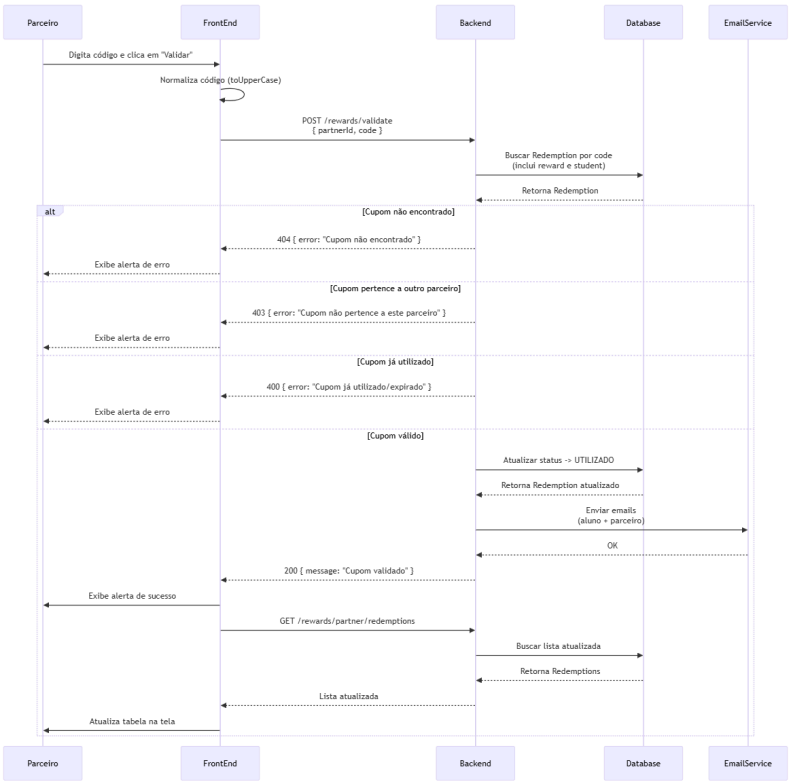
Sessao / TokenAutenticacao (se modelado)

- Atributos: id, usuarioId, token, dataCriacao, dataExpiracao.

3.4 Diagramas de Sequência


Professor (navegador)
Front-end React (ProfessorDashboard)
Backend Express (/transactions/transfer)
Prisma + PostgreSQL
Serviço de E-mail (Mailer / SMTP)
Notifications (tabela Notification)
Aluno (navegador)
Preenche: aluno, quantidade, motivo e clica 'Enviar moedas'
UI usuário logado (AuthContext) pega backendId do professor e token JWT
POST /transactions/transfer Authorization: Bearer {JWT} Body: { professorId, alunoId, amount, reason }
#Middleware authenticateToken valida JWT e cria PROFESSOR
Buscar professor por id + incluir account
Professor + conta
Buscar aluno por id + incluir account
Aluno + conta
Verifica se professor.role = PROFESSOR e aluno.role = ALUNO
Verifica saldo do professor >= amount
alt
[Saldo insuficiente]
400 { error: 'Saldo insuficiente' }
Exibe alerta de erro
[Saldo suficiente]
Inicia prisma.$transaction(...)
Decrementa saldo da conta do professor
Incrementa saldo da conta do aluno
Cria Transaction type=ENVIO_PROFESSOR_ALUNO description=reason
Retorna contas atualizadas + transaction
INSERT em Notification userId = aluno.id title = 'Você recebeu moedas' read = false
Notificação criada
sendStudentReceivedCoinsEmail({ alunoName, alunoEmail, professorName, amount, reason })
OK (ou erro logado, mas sem quebrar a resposta)
201 { updatedProfessorAcc, updatedAlunoAcc, transaction }
Atualiza saldo do professor na tela Exibe mensagem de sucesso
Mostra feedback 'Moedas enviadas com sucesso'
Abre StudentDashboard
GET /accounts/{studentId}/summary
Carrega account + transações
Dados do saldo + histórico
Saldo + transactions (inclui recebimento)
Mostra novo saldo e extrato (linha 'Recebeu X moedas do professor Y')
Clica no sino 🔔
GET /notifications?userId=alunoId
SELECT * FROM Notification WHERE userId = alunoId
Lista de notificações (incluindo a de moedas)
JSON com notificações
Mostra modal de notificações com aviso 'Você recebeu X moedas'


Aluno
FrontEnd
Backend
Database
EmailService
Clica em "Resgatar" em uma vantagem
Verifica se saldo exibido >= cost
alt
[Saldo insuficiente (lado do front)]
Exibe alerta "Saldo insuficiente"
[Saldo suficiente]
POST /rewards/redeem\n{ studentId, rewardId }
Buscar aluno por ID (inclui account)
Retorna aluno + account
Buscar reward por ID\n(inclui partner)
Retorna reward + partner
alt
[Aluno inválido ou reward inativa]
400 { error: ... }
Exibe alerta de erro
[Saldo insuficiente (lado do backend)]
400 { error: "Saldo insuficiente" }
Exibe alerta de erro
[Tudo válido]
Início de transação
Debita saldo da conta do aluno
Conta atualizada
Cria Redemption\n(status = GERADO, code = COUPON)
Retorna Redemption
Cria Transaction\n(type = RESGATE_VANTAGEM,\n couponCode = COUPON)
Retorna Transaction
Commit da transação
Enviar emails de resgate\n(aluno + parceiro,\n com rewardTitle e couponCode)
OK (ou loga erro se falhar)
201 { couponCode, updatedAccount, transaction }
Atualiza saldo e extrato\n(com transaction)
Exibe alerta de sucesso\n+ mostra código do cupom



Fluxo – Cadastro de Aluno (UC-01)

1. Aluno acessa a tela de cadastro.
2. Frontend envia requisição `POST /alunos` ao backend com os dados preenchidos.
3. Backend valida campos, verifica se instituição existe e se e-mail/CPF não estão em uso.
4. Backend cria registro do aluno no banco e dispara solicitação para o serviço de e-mail.
5. Serviço de e-mail envia mensagem de confirmação.
6. Backend retorna resposta de sucesso ao frontend, que exibe mensagem ao usuário.

Fluxo – Envio de Moedas (UC-05)

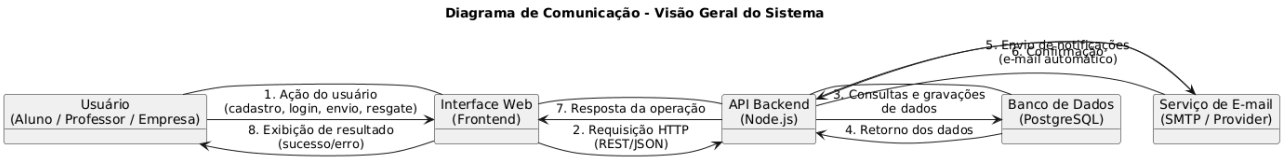
1. Professor autenticado abre tela de envio de moedas.
2. Frontend envia requisição `POST /transacoes/enviar` com professorId, alunoId, valor, motivo.

3. Backend verifica saldo do professor e existência do aluno.
4. Backend cria registro de transação, atualiza os saldos de professor e aluno.
5. Backend aciona serviço de e-mail para notificar o aluno.
6. Frontend recebe confirmação e exibe mensagem de sucesso.

Fluxo – Resgate de Vantagem (UC-07)

1. Aluno autenticado acessa catálogo de vantagens e escolhe uma opção.
2. Frontend envia `POST /resgates` com alunoId e vantagemId.
3. Backend verifica saldo do aluno, status da vantagem e custo em moedas.
4. Backend registra a transação de resgate, atualiza o saldo do aluno e cria um cupom.
5. Backend envia solicitação ao serviço de e-mail para notificar aluno e empresa com o código do cupom.
6. Frontend recebe resposta de sucesso e exibe o cupom (ou sua confirmação) na tela.

3.5 Diagramas de Comunicação



Os diagramas de comunicação, conceitualmente, envolvem os mesmos participantes dos fluxos de sequência, enfatizando **quem troca mensagens com quem**:

- No envio de moedas:
  - Professor ↔ Frontend ↔ Backend ↔ Banco de Dados
  - Backend ↔ Serviço de E-mail
- No resgate de vantagem:

- Aluno ↔ Frontend ↔ Backend ↔ Banco de Dados
- Backend ↔ Serviço de E-mail ↔ Empresa Parceira

A ênfase aqui é na **estrutura das mensagens** (requisições HTTP, comandos ao banco, chamadas ao serviço de e-mail) e nos relacionamentos entre objetos que cooperam para completar cada caso de uso.

3.6 Diagramas de Estados

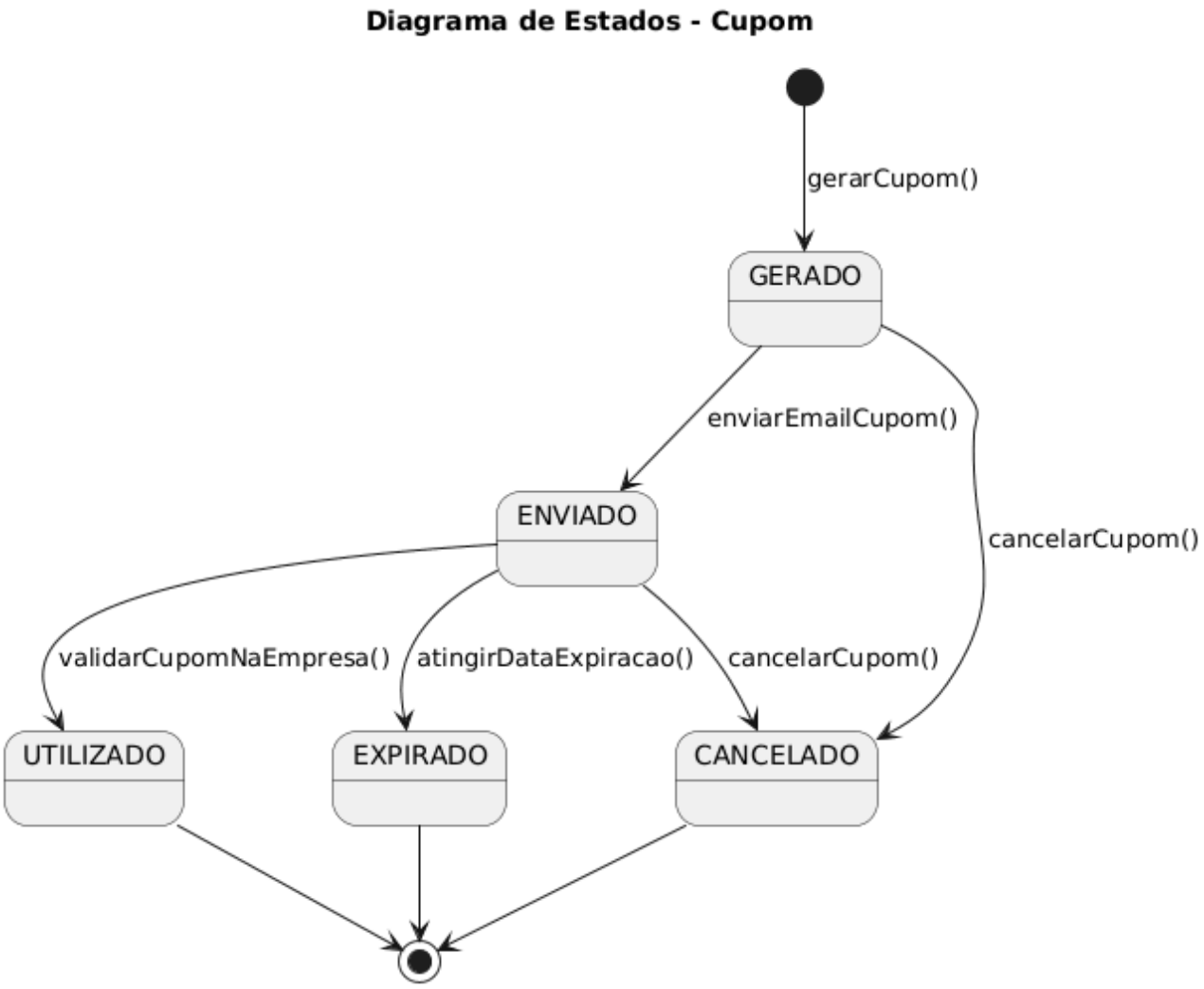


Diagrama de Estados - Vantagem (Produto/Desconto)

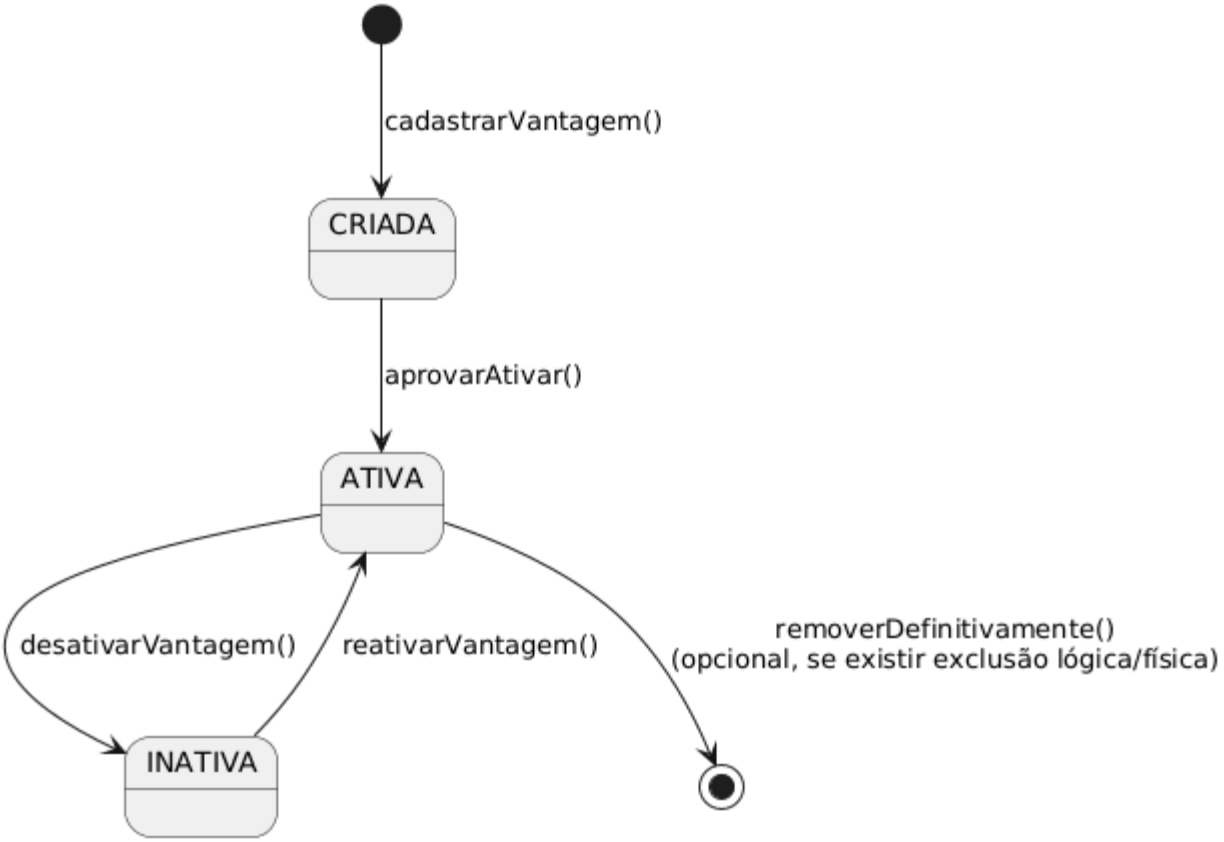
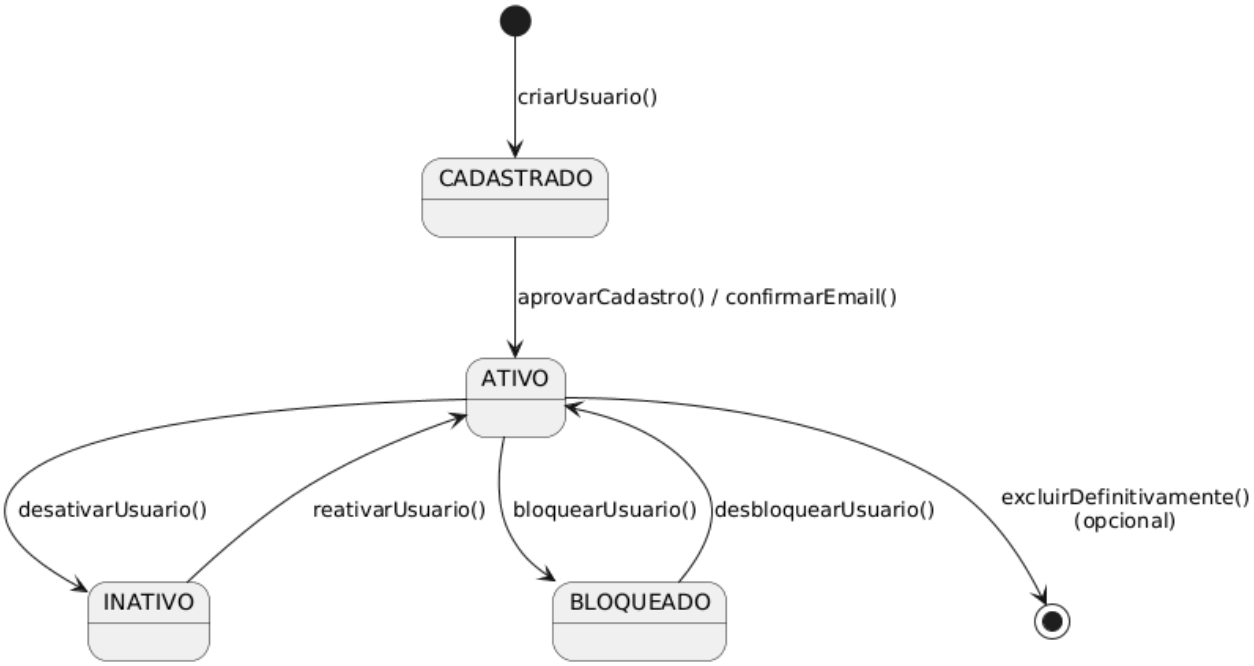


Diagrama de Estados - Usuário



Algumas entidades do sistema possuem ciclos de vida bem definidos:

Cupom

- *Estados típicos:*
    - *GERADO → ENVIADO → UTILIZADO*
    - *GERADO → ENVIADO → EXPIRADO*
    - *GERADO → CANCELADO*
- *Transições:*
    - *“Gerar cupom” ao concluir resgate.*
    - *“Enviar cupom” ao disparar e-mail.*
    - *“Utilizar cupom” quando a empresa valida o código.*
    - *“Expirar cupom” após data limite.*
    - *“Cancelar cupom” em casos de erro ou fraude.*

Vantagem

- *Estados: CRIADA → ATIVA → INATIVA.*
- *Transições:*
    - *Criação pela empresa.*
    - *Ativação para torná-la visível aos alunos.*
    - *Inativação para retirá-la temporariamente do catálogo.*

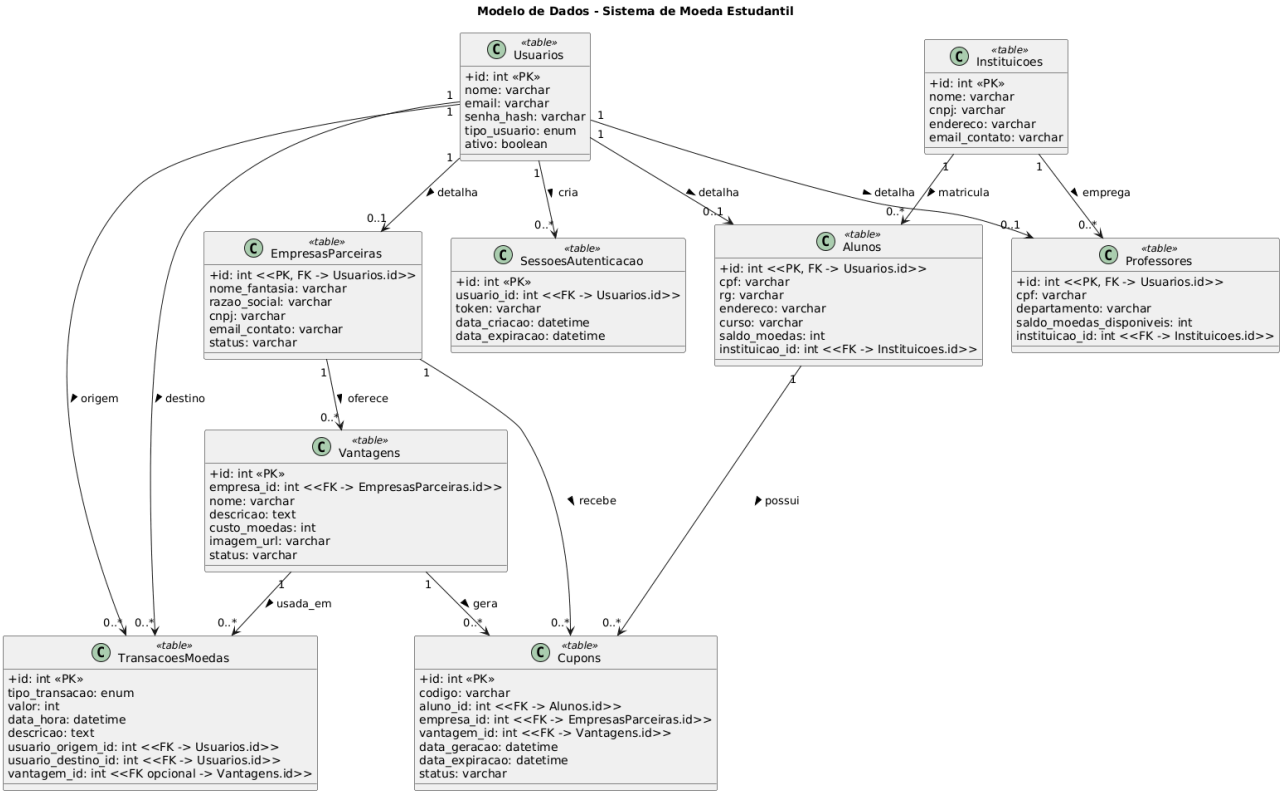
Usuário (Empresa / Aluno / Professor)

4. *Estados: CADASTRADO → ATIVO → INATIVO/BLOQUEADO.*
5. *Transições:*

5.1 *Ativação após cadastro/validação.*

5.2 *Inativação ou bloqueio por decisão administrativa ou problemas de segurança.*

6. Modelos de Dados



A seguir é apresentada uma visão conceitual das principais tabelas do banco de dados e suas relações:

Tabela `usuarios`

- id (PK)
- nome
- email (único)
- senha_hash
- tipo_usuario (ALUNO, PROFESSOR, EMPRESA, INSTITUICAO, ADMIN)
- ativo (boolean)

Tabela `alunos`

- id (PK, FK → usuarios.id)
- cpf (único)
- rg
- endereco
- curso
- instituicao_id (FK → instituicoes.id)
- saldo_moedas

Tabela `professores`

- id (PK, FK → usuarios.id)
- cpf (único)
- departamento
- instituicao_id (FK → instituicoes.id)
- saldo_moedas_disponiveis

Tabela `instituicoes`

- id (PK)
- nome
- cnpj (opcional)
- endereco
- email_contato

Tabela `empresas_parceiras`

- id (PK, FK → usuarios.id ou tabela própria associada a usuarios)
- nome_fantasia
- razao_social
- cnpj (único)
- email_contato
- status (ATIVA/INATIVA)

Tabela `vantagens`

- id (PK)
- empresa_id (FK → empresas_parceiras.id)
- nome
- descricao
- custo_moedas
- imagem_url
- status (ATIVA/INATIVA)

Tabela `transacoes_moedas`

- id (PK)
- tipo_transacao (ENVIO_PROF_ALUNO, RESGATE, AJUSTE, etc.)
- valor
- data_hora
- usuario_origem_id (FK → usuarios.id, pode ser NULL em alguns tipos)
- usuario_destino_id (FK → usuarios.id)

- descricao
- vantagem_id (FK opcional → vantagens.id, quando for resgate)

Tabela `cupons`

- id (PK)
- codigo (único)
- aluno_id (FK → alunos.id)
- empresa_id (FK → empresas_parceiras.id)
- vantagem_id (FK → vantagens.id)
- data_geracao
- data_expiracao (opcional)
- status (GERADO, ENVIADO, UTILIZADO, EXPIRADO, CANCELADO)

Tabela `sessoes` (ou tokens de autenticação)

- id (PK)
- usuario_id (FK → usuarios.id)
- token
- data_criacao
- data_expiracao

Estratégia de Mapeamento Objeto–Relacional

- Cada **entidade de domínio** (Aluno, Professor, Instituição, Empresa, Vantagem, Transação, Cupom) é mapeada para uma tabela correspondente ou combinação de tabelas (`usuarios` + tabela específica).

- O ORM (ex.: Prisma) é responsável por:
    - Gerar o esquema do banco.
    - Executar migrações.
    - Realizar consultas e persistência via modelos de alto nível.
- As relações 1:N e N:1 (por exemplo, Instituição–Professor, Empresa–Vantagem, Aluno–Transação) são representadas com **chaves estrangeiras** e são navegadas pelo código através de propriedades de relação fornecidas pelo ORM.