ads-praticas-extensionistas-3-2025

Participantes

> Luis Carlos Becker (conta.becker@gmail.com)

> Paulo Luis Hammes (paulolb03@gmail.com)

> Vitor Mateus Weirich (weirichvitor@gmail.com)

Projeto

- Um sistema WEB de doações visando apoiar a comunidade.

Com os requisitos básicos de:

1. Cadastro de usuários (doadores e admins)
2. Um 'portal' para visualizar as campanhas de doação em curso
3. Um página de transparência (onde está sendo alocados as doações)

Link do gitHub

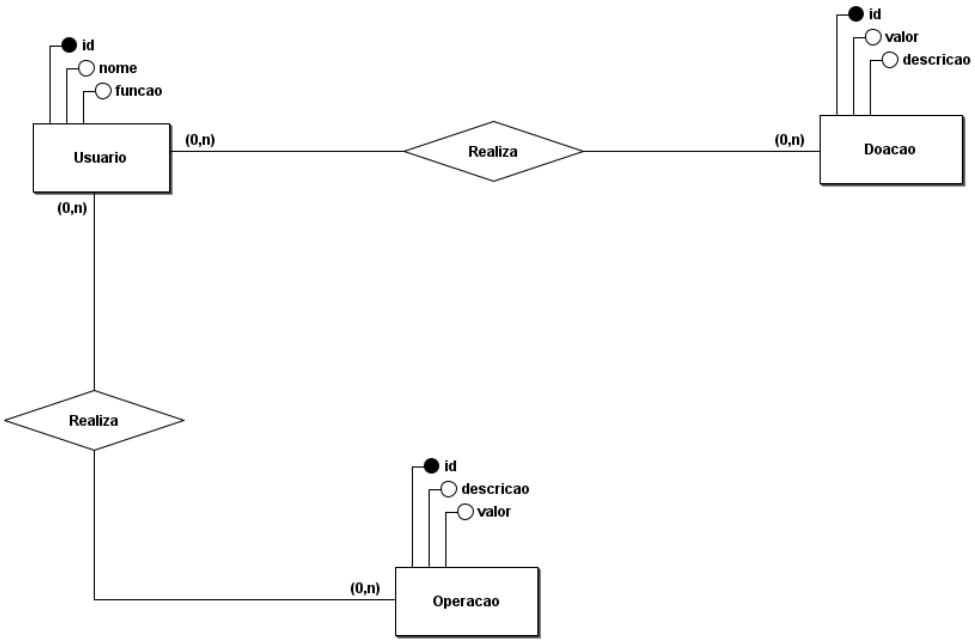
ads-praticas-extensionistas-3-2025

Link do projeto

Demo pública (aplicação frontend): `https://doacoes.vitorweirich.com/`

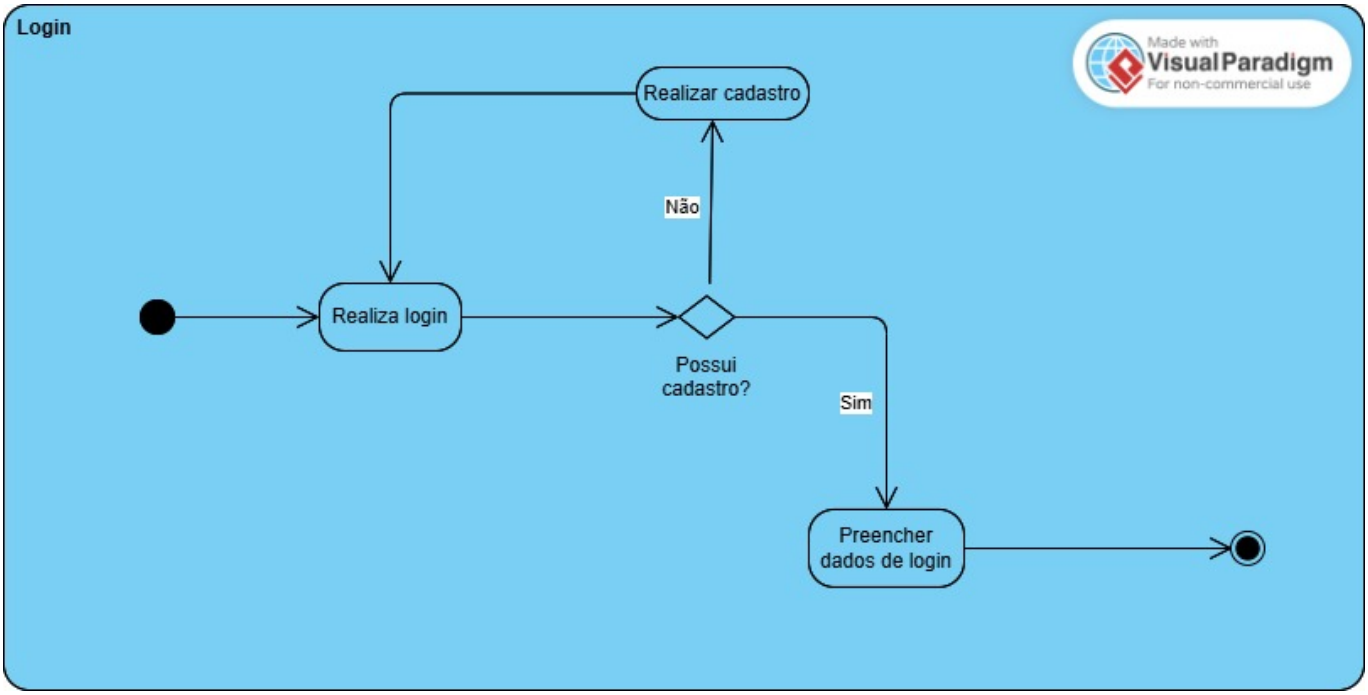
Diagramas

Entidade Relacionamento



Atividade

Realizar login



Realizar operação

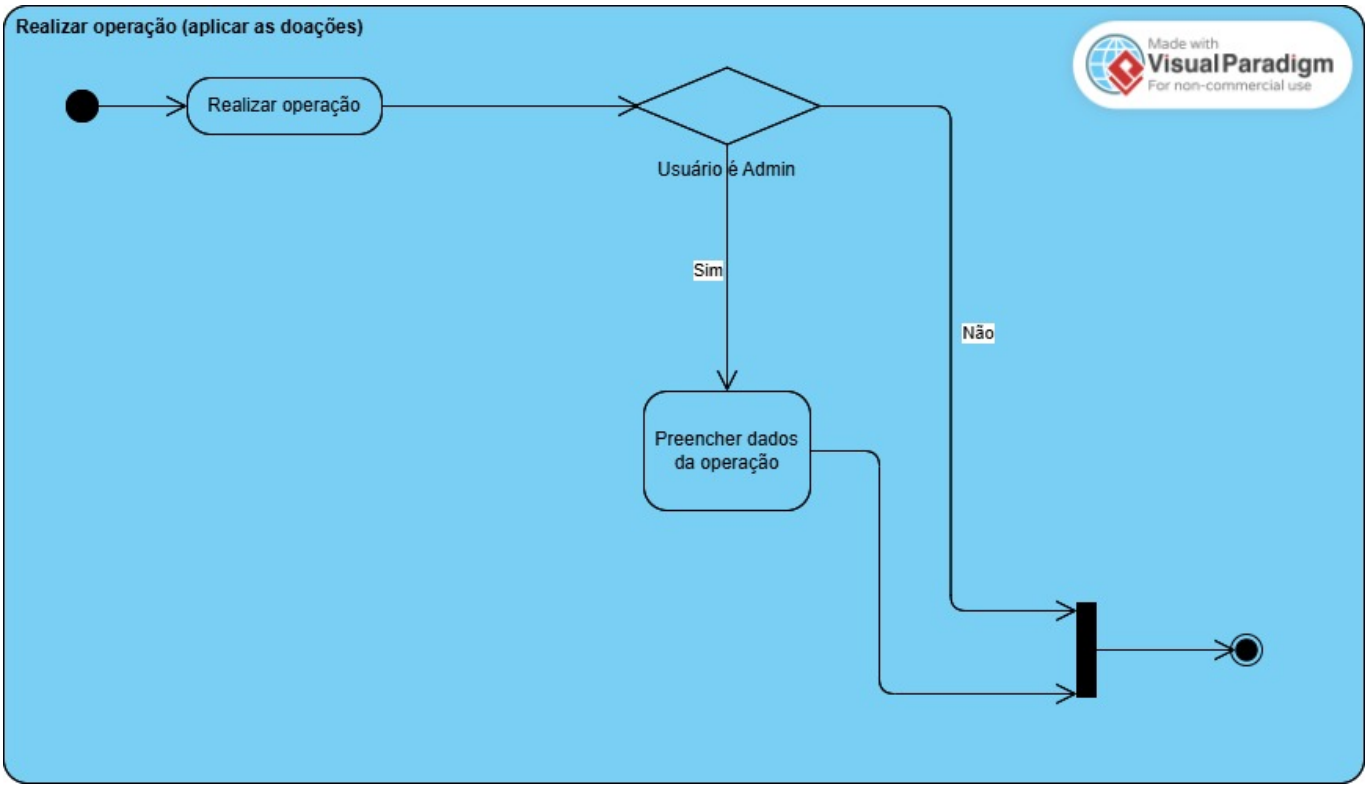


Diagrama de caso de uso

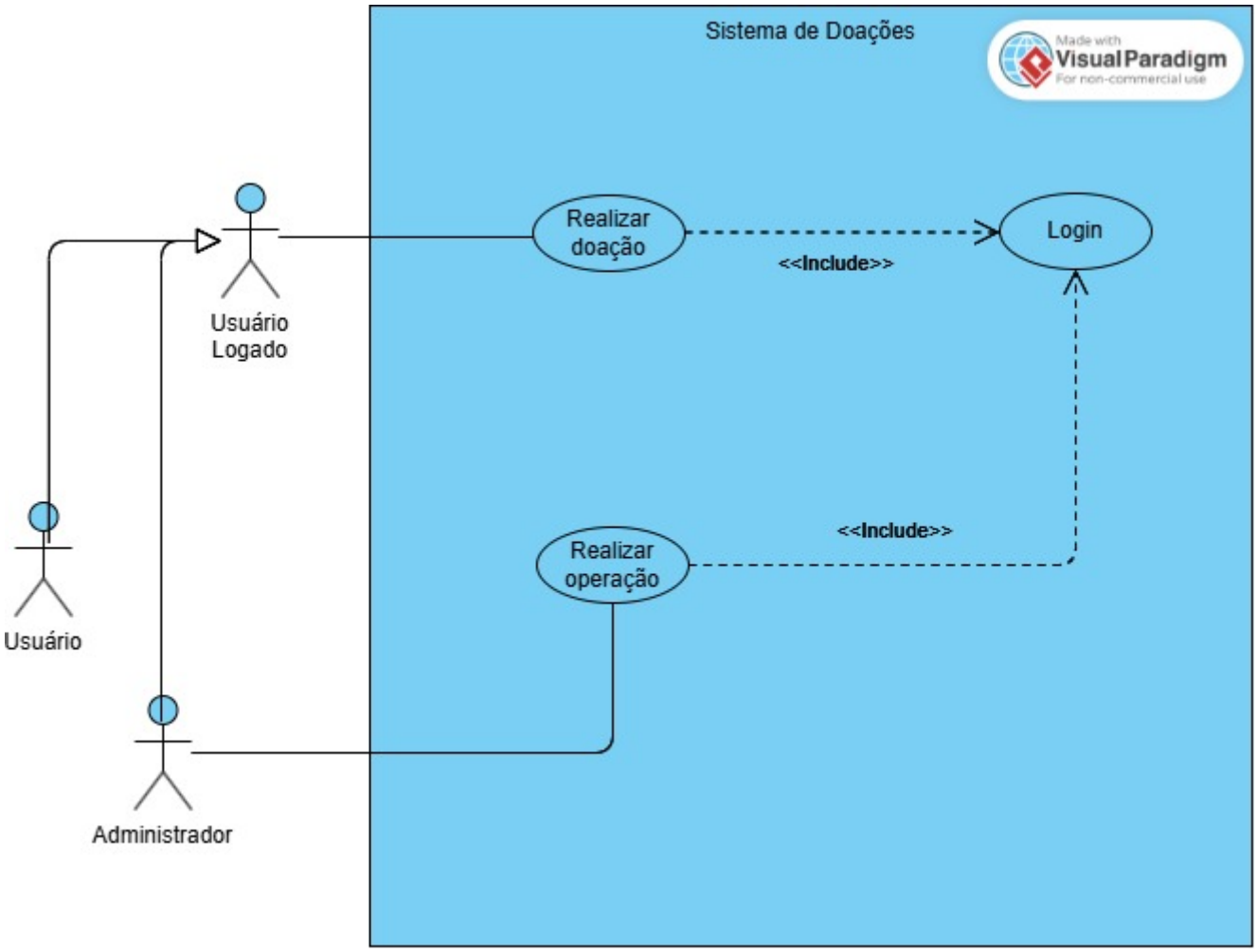
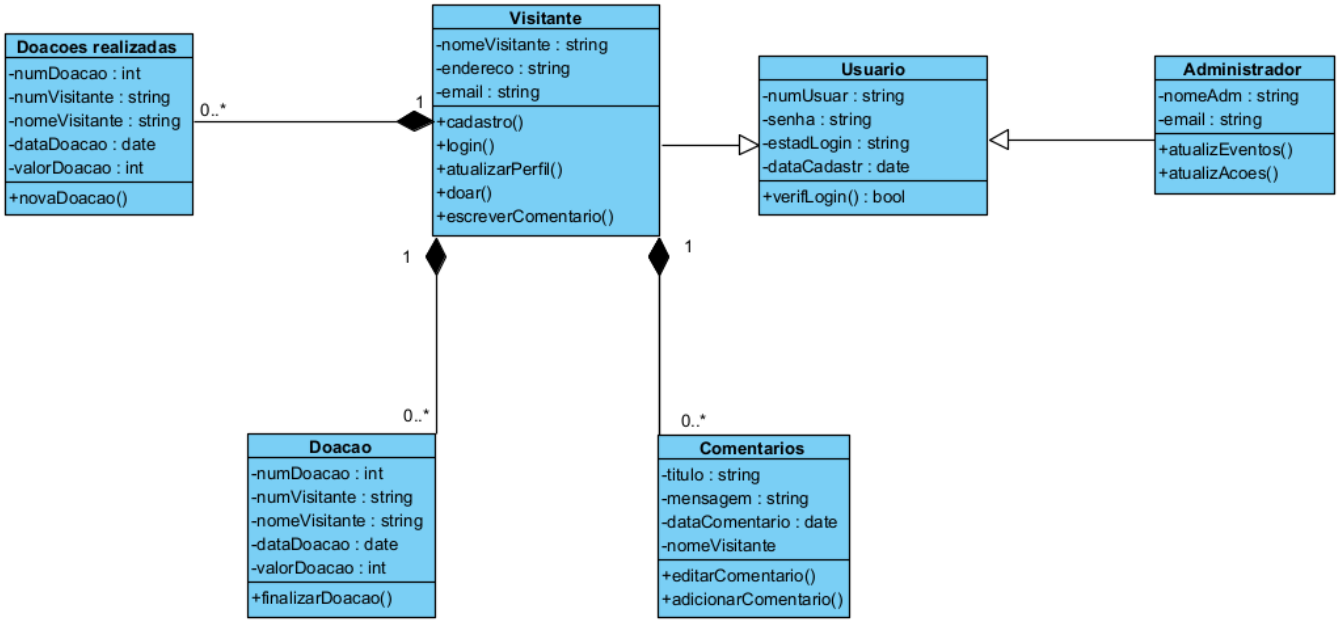
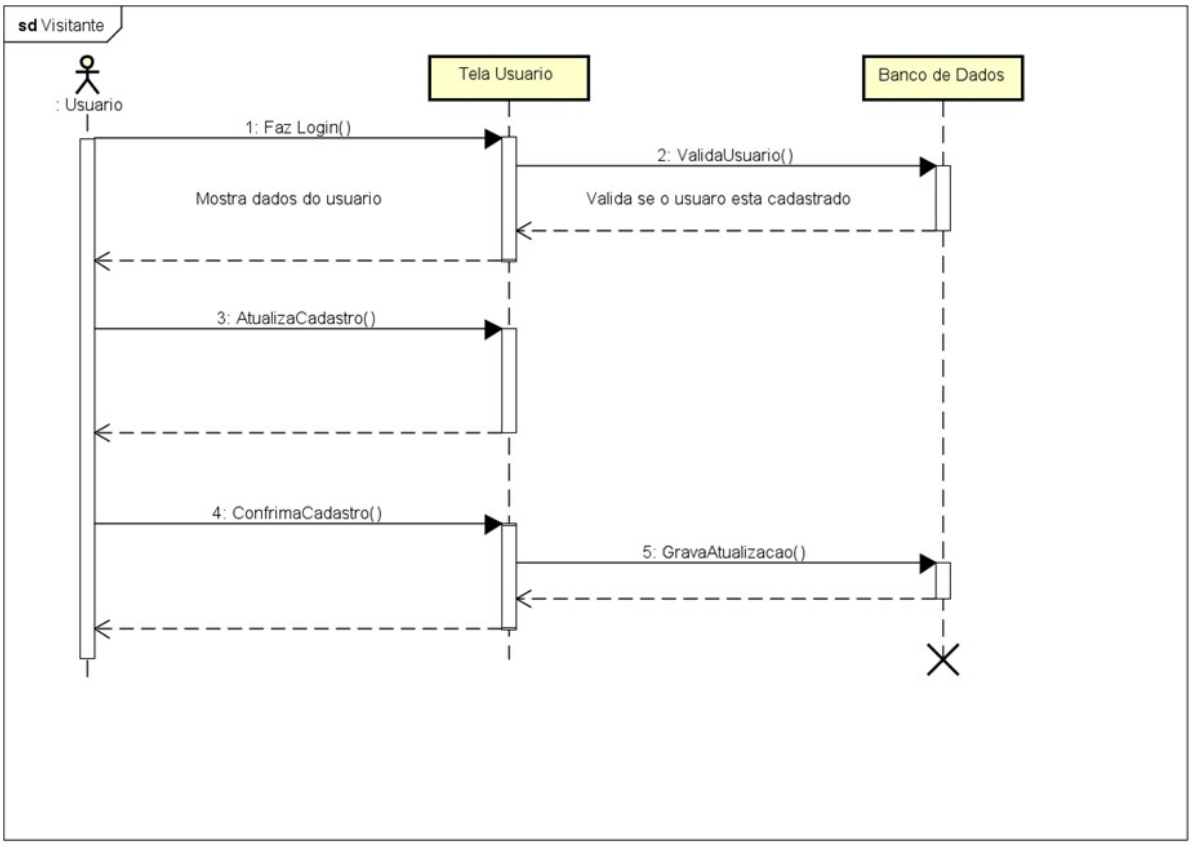


Diagrama de classes

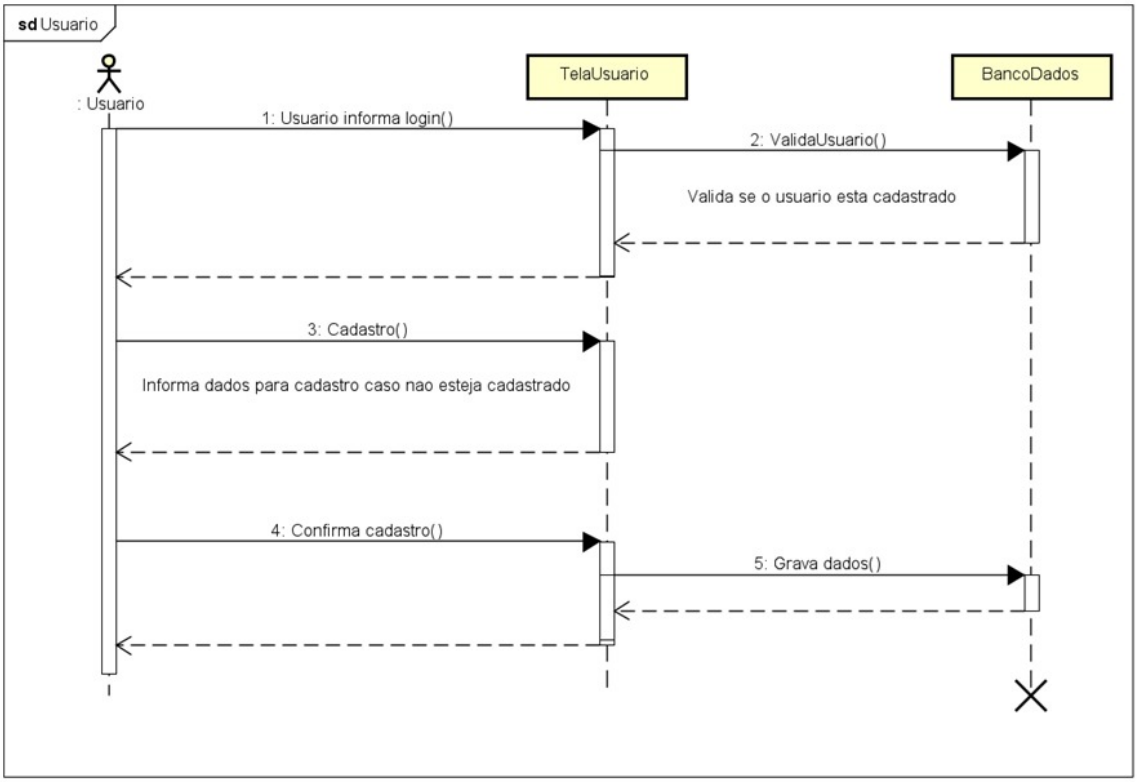


Diagramas de sequência

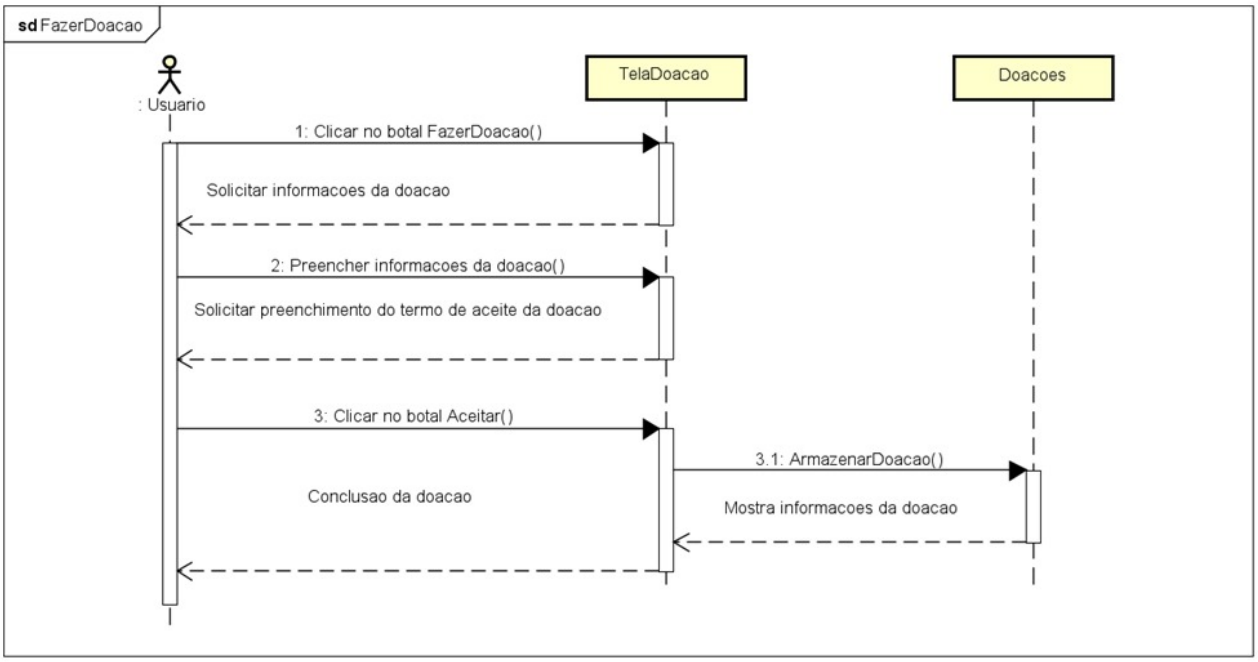
Visitante



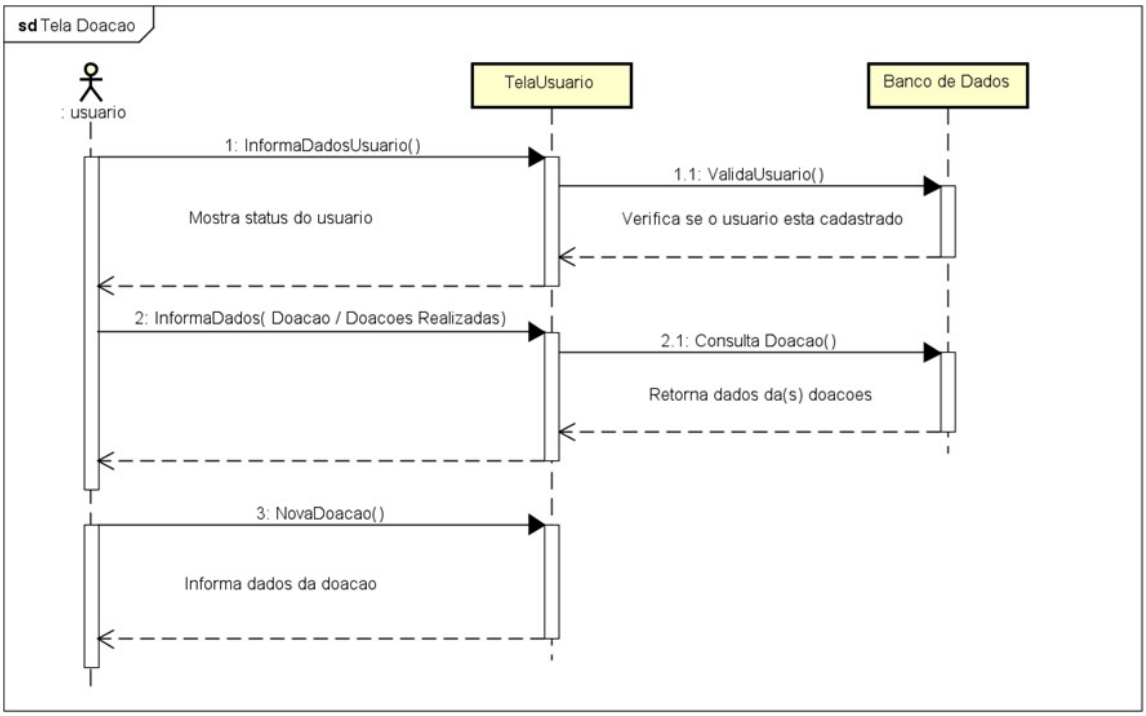
Usuario



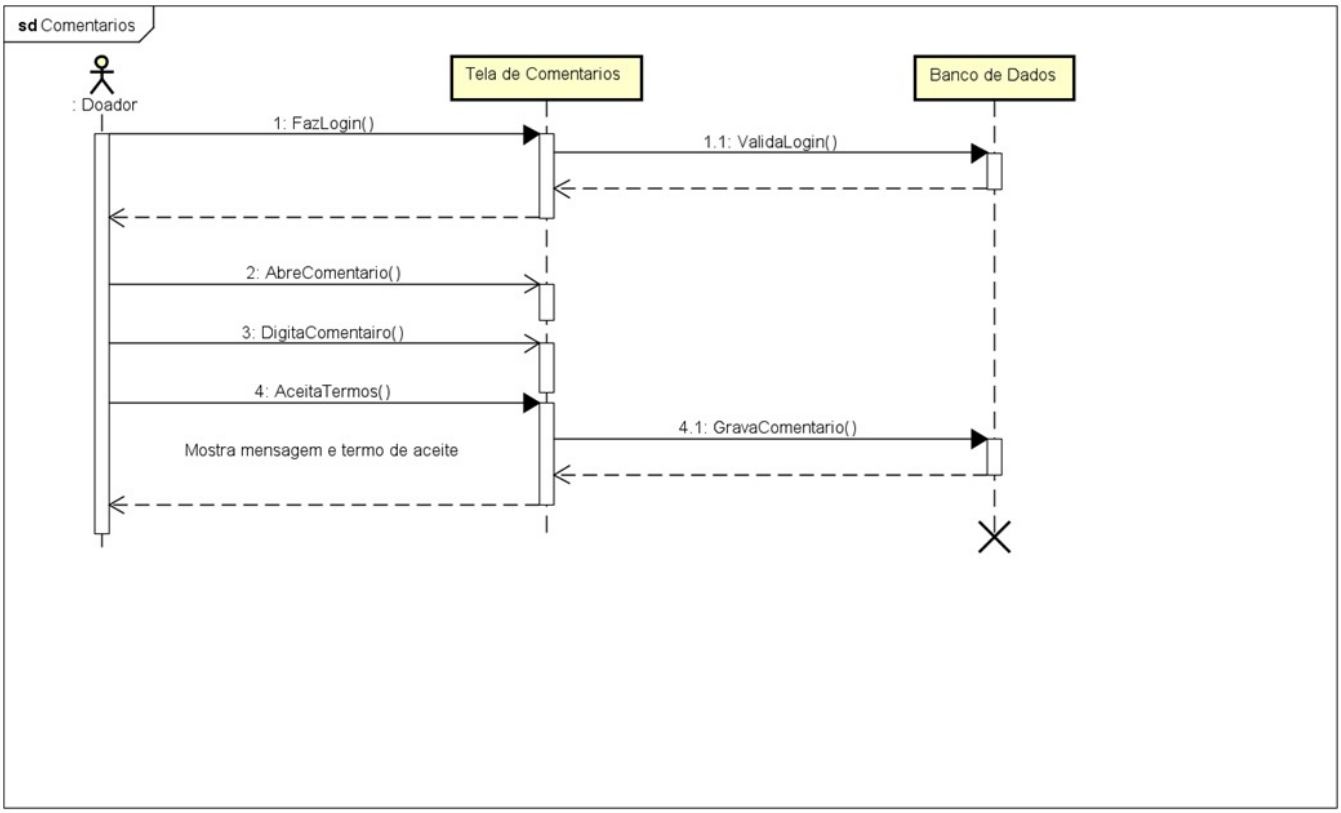
Fazer Doação



Tela de Doações



Comentários

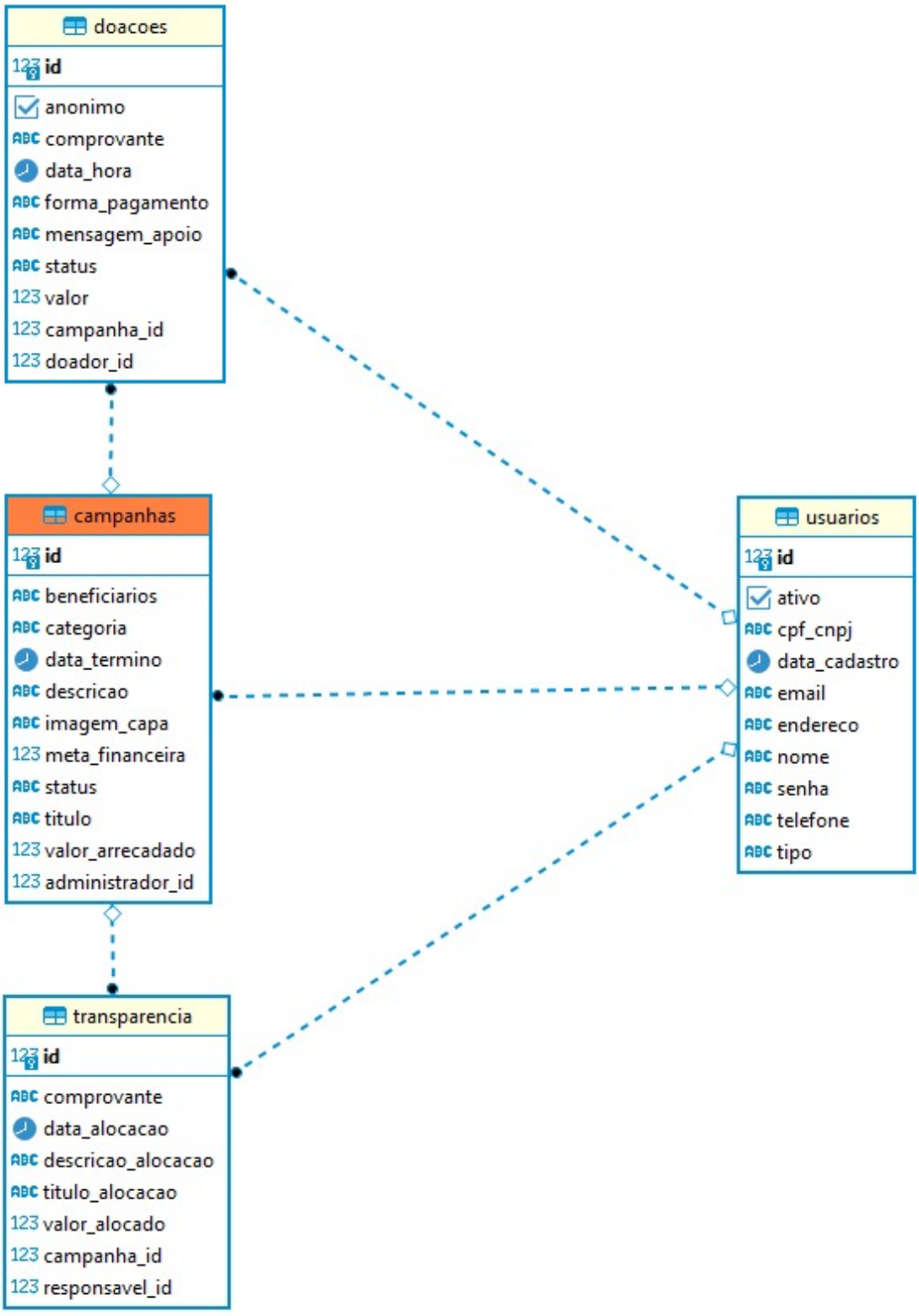


Modelagem e DDL do Banco de Dados

A modelagem do banco de dados foi realizada com base nos requisitos do sistema, utilizando o modelo entidade-relacionamento para definir as principais entidades e seus relacionamentos. A estrutura do banco foi implementada em PostgreSQL, e os scripts de criação das tabelas e relacionamentos estão disponíveis na pasta `./modelagem-banco-de-dados`.

Arquivos disponíveis:

- `modelagem-banco-de-dados/DDL.sql`: Script de criação das tabelas e constraints do banco.
- `modelagem-banco-de-dados/init.script.sql`: Script de inicialização e inserção de dados exemplo.
- `modelagem-banco-de-dados/relacionamento-entidades.jpeg`: Imagem do diagrama de relacionamento das entidades.



Codigo Fonte do sistema

O sistema possui uma arquitetura cliente-servidor, com backend em Java Spring Boot, banco de dados PostgreSQL e frontend em Vue.js. Para mais detalhes veja o README.md no caminho `./sistema-doacoes-completo/README.md`, juntamente como todo o codigo fonte que é separado em pastas.

- Frontend: `./sistema-doacoes-completo/frontend`
- Backend: `./sistema-doacoes-completo/backend`
- Demo (aplicação frontend): `https://doacoes.vitorweirich.com/`

Práticas Extensionistas IV

Documentação específica da disciplina (diagramas de arquitetura, DevOps, escolha de infraestrutura e justificativas) está consolidada em:

`./praticas-IV/README.MD`

Link direto no GitHub: Práticas Extensionistas IV – Entrega 1

Conteúdos presentes:

- Diagrama de Pacotes (Arquitetura da Aplicação)
- Diagrama de Implantação
- Diagrama de Arquitetura DevOps (CI/CD)
- Justificativa da infraestrutura (Self-host em VPS com Coolify/Dokploy) e plano de evolução

Guia de Deploy

Para instruções detalhadas de deploy do sistema em VPS com Coolify, consulte o arquivo `DEPLOY.md` na raiz do repositório.

> **Assista à demonstração em vídeo do processo de deploy:**youtube

---

Workflow de Deploy (GitHub Actions)

Este repositório contém um workflow de deploy automatizado via GitHub Actions em `.github/workflows/build-on-pr-merge.yml`.

Resumo do funcionamento:

- Disparo: a cada push na branch `master` cujo commit contenha `#deploy` na mensagem.
- Build: gera e publica imagens Docker do backend e do frontend para o registro definido em `REGISTRY`.
- Deploy: aciona webhooks do Coolify (backend e frontend) se os segredos estiverem configurados.

Detalhes importantes:

1. Registro de contêiner
    - Variável de ambiente: `REGISTRY` (padrão: `registry.vitorweirich.com`).
    - Login usa `secrets.REGISTRY_USERNAME` e `secrets.REGISTRY_PASSWORD`. Caso não estejam definidos, cai no fallback `github.actor` e `secrets.TOKEN`.
2. Imagens geradas
    - Backend: `${REGISTRY}/${IMAGE_BACKEND}:latest` e `${REGISTRY}/${IMAGE_BACKEND}:${{ github.sha }}`
    - Frontend: `${REGISTRY}/${IMAGE_FRONTEND}:latest` e `${REGISTRY}/${IMAGE_FRONTEND}:${{ github.sha }}`
    - Plataformas: `linux/amd64` (ajustável no workflow se precisar multi-arch).
3. Deploy via Coolify

- Se os webhooks estiverem definidos, o workflow faz `GET` nas URLs de webhook com ou sem `Authorization: Bearer` (quando `COOLIFY_TOKEN` está presente).
- Segredos esperados: `COOLIFY_BACKEND_WEBHOOK`, `COOLIFY_FRONTEND_WEBHOOK`, `COOLIFY_TOKEN` (opcional).

4. Build do frontend
   - Passa `build-args: MODE=production` para o `docker buildx` do frontend.

Como acionar o deploy

1. Faça merge ou push na branch `master` com a mensagem de commit contendo `#deploy`.

Exemplos de mensagens:

```
feat: ajustar layout da home #deploy
chore(release): v1.2.3 #deploy
```

2. Aguarde a execução do workflow no GitHub Actions (aba Actions do repositório).

Segredos necessários (Settings → Secrets and variables → Actions)

- `REGISTRY_USERNAME` e `REGISTRY_PASSWORD`: credenciais do seu registro de contêiner.
- `TOKEN`: opcional (fallback para autenticação quando não usar `REGISTRY_*`).
- `COOLIFY_BACKEND_WEBHOOK` e `COOLIFY_FRONTEND_WEBHOOK`: URLs de deploy no Coolify.
- `COOLIFY_TOKEN`: opcional, caso o webhook exija Bearer Token.

Variáveis de ambiente do workflow (podem ser ajustadas no arquivo do workflow)

- `REGISTRY`: endereço do registro Docker.
- `IMAGE_BACKEND`: nome do repositório da imagem do backend.
- `IMAGE_FRONTEND`: nome do repositório da imagem do frontend.

Documentação da API Pública

Para integração externa e consulta das campanhas públicas, consulte a documentação simplificada em `PUBLIC_API.md`.