CENTRO UNIVERSITÁRIO CATÓLICA DE SANTA CATARINA

ENGENHARIA DE SOFTWARE

LEONARDO PEREIRA BORGES

Plataforma de Crowdfunding Simplificada — RFC v2

Joinville

2025

Resumo

Esta RFC v2 atualiza e consolida a especificação da plataforma web de crowdfunding, alinhando o documento ao estado atual do repositório. A solução é composta por SPA em Quasar/Vue 3 (TypeScript) e backend Node.js 20 + Express, com PostgreSQL (Prisma), autenticação via Clerk, pagamentos via Stripe (Checkout e Connect), armazenamento de mídias em S3, SSE para eventos simples e endpoint de métricas de plataforma. O projeto aplica boas práticas de segurança (CSP, rate limit, idempotência), qualidade (testes e cobertura), documentação (OpenAPI) e observabilidade (New Relic).

1. Introdução

1.1 Contexto

O crowdfunding democratiza o financiamento de projetos por meio de contribuições da comunidade. Com o avanço da web, tornou-se viável construir plataformas seguras e escaláveis que conectam criadores e apoiadores em uma experiência confiável e transparente.

1.2 Justificativa

O projeto permite aplicar fundamentos práticos de Engenharia de Software: arquitetura monolítica modular, segurança de aplicação e dados (LGPD), modelagem relacional consistente, integração com serviços externos (Clerk, Stripe, AWS S3), automação de testes e entrega contínua.

1.3 Objetivos

- Objetivo principal: disponibilizar uma plataforma completa para cadastrar, descobrir e apoiar campanhas com segurança e boa experiência de uso.
- Objetivos específicos:
    - Autenticação e gestão de sessão com Clerk.
    - UI responsiva moderna com Quasar/Vue 3 (TypeScript).
    - API RESTful com Express + Prisma e validações robustas.
    - Processamento de pagamentos via Stripe Checkout e suporte a Connect/assinaturas.
    - Recursos sociais básicos (comentários) e gestão de mídias (imagens/vídeos).
    - Segurança: rate limiting, idempotência, CSP, logs estruturados e LGPD.
    - Documentação OpenAPI, testes e integração com observabilidade.

2. Descrição do Projeto

2.1 Propósito e uso prático

- Criadores publicam campanhas com metas de arrecadação e prazos.
- Apoiadores descobrem campanhas, contribuem com valores seguros via Stripe e acompanham o progresso.

- A comunidade interage via comentários, aumentando transparência e engajamento.

2.2 Público-alvo

- Criadores (empreendedores, artistas, ONGs), Apoiadores (público geral), 18–55 anos.

2.3 Problemas endereçados

- Visibilidade de projetos pequenos, barreiras técnicas para campanhas, transparência e confiança nas transações.

2.4 Diferenciais

- UX simplificada, integração nativa com Clerk/Stripe, transparência (comentários/métricas), performance e documentação aberta.

3. Especificação Técnica

3.1 Arquitetura e Stack

- Monólito: Backend Node.js 20 + Express; Frontend Quasar/Vue 3 (TypeScript).
- Banco: PostgreSQL com Prisma ORM.
- Autenticação: Clerk (Express middleware no backend; integração no frontend).
- Pagamentos: Stripe (Checkout sessions; webhooks; Connect; assinaturas).
- Mídias: AWS S3 como storage principal (opcional `ASSETS_BASE_URL` /CDN).
- Eventos e métricas: SSE público e endpoint de métricas agregadas.

Arquivos-chave:

- Backend: `backend/src/app.ts`, `backend/src/middleware/*.ts`, `backend/src/controllers/*.ts`, `backend/src/routes/*.ts`, `backend/prisma/schema.prisma`.
- Frontend: `frontend/src/**/*`, `frontend/quasar.config.ts`.
- Documentação: `openapi.json` (raiz).

3.2 Segurança

- CORS com allowlist a partir de `FRONTEND_ORIGIN`.
- CSP via `helmet` com domínios para Stripe/Clerk e mídia estática (inclui `getPublicBaseUrl()` do S3).
- Rate limiting:
  - Global: 300 req/15 min por IP/usuário.
  - Criação de projeto: 10 req/min.
  - Criação de retirada (reserva/futuro): 1/dia.
- Idempotência: suporte a header `Idempotency-Key` em `POST/PUT/PATCH`, persistido em `IdempotencyKey` (Prisma).
- Identidade de requisição: `X-Request-Id` gerado e retornado; logs estruturados com IP anonimizado e redaction de PII.
- Respostas autenticadas com `Cache-Control: no-store`.
- HTTPS opcional com redirecionamento 308 baseado em `ENFORCE_HTTPS` + `CANONICAL_BASE_URL`.

3.3 Autenticação (Clerk)

- Middleware do Clerk habilitado apenas quando configurado (`CLERK_SECRET_KEY` ou `CLERK_JWT_VERIFICATION_KEY`).
- Extração de usuário por `getAuth(req)`; `userId` propagado em `req`.
- Sincronização de perfil no primeiro acesso (upsert de `User` – nome/email quando disponível).
- Bypass de autenticação para testes controlado por variáveis/headers apenas em `NODE_ENV=test`.

3.4 Pagamentos (Stripe)

- Contribuições únicas via Stripe Checkout Session (mínimo R$ 5,00). Metadados incluem `projectId` e, quando presente, `userId`.
- Webhook em `/api/webhooks/stripe` com corpo raw e verificação de assinatura (`STRIPE_WEBHOOK_SECRET`).
- Connect (onboarding/repasse) e assinaturas (modelo `Subscription`) disponíveis na base do projeto.
- No texto: "Stripe Checkout" é a implementação padrão; "Payment Links" pode ser citado como alternativa futura.

3.5 Mídias (Imagens e Vídeos)

- Imagens de campanha em S3: validação de MIME/tamanho (máx. 5MB), persistência de metadados e ordenação; remoção segura com limpeza no storage quando possível.
- Vídeos: rotas dedicadas para upload/gestão.
- Compatibilidade local mantida em `backend/uploads/projects` (quando sem S3).

3.6 Eventos e Métricas

- SSE público em `GET /api/events` para eventos simples (sem auth obrigatória; aceita hints de usuário por headers quando presentes).
- Métricas de plataforma em `GET /api/stats/platform` (total de projetos, total arrecadado, apoiadores únicos, taxa de sucesso).

3.7 Modelagem de Dados (Prisma)

Modelos principais em `backend/prisma/schema.prisma`:

- `User`, `Category`, `Project`, `ProjectImage`, `Contribution`, `Comment`, `Subscription`, `IdempotencyKey`.
- Enums: `ProjectStatus`, `FundingType`, `SubscriptionInterval`, `SubscriptionStatus`.
- Atualizações de `Project`: campos para recorrência (`subscriptionEnabled`, `subscriptionPriceCents`, `subscriptionInterval`), métricas (`raisedCents`, `supportersCount`) e relacionamento `images`.

3.8 API e Endpoints

- Fonte de verdade: `openapi.json` (raiz).
- Rotas principais (prefixo `/api`): `projects`, `project-images`, `project-videos`, `contributions`, `subscriptions`, `comments`, `checkout`, `webhook`, `categories`, `me`, `connect`, `events`.
- Destaques:
  - `POST /api/checkout/session` — cria sessão de pagamento (mín. R$ 5,00).
  - `POST /api/webhooks/stripe` — webhook Stripe com body raw.
  - `GET /api/events` — SSE público.
  - `GET /api/stats/platform` — métricas agregadas.

4. Requisitos

4.1 Funcionais (exemplos)

- Cadastro e autenticação de usuários via Clerk.
- CRUD de campanhas e listagem pública com filtros.
- Visualização detalhada da campanha com progresso e comentários.
- Contribuição financeira via Stripe Checkout; registro de contribuições e status.
- Upload/gestão de imagens (máx. 5 por campanha) e vídeos.

- Painel pessoal (opcional) e assinaturas recorrentes (quando habilitadas na campanha).

4.2 Não Funcionais

- Disponibilidade alvo ≥ 99,5%.
- Performance SPA: carregamento inicial < 2s (ambiente típico).
- Cobertura de testes: alvo ≥ 80%.
- Segurança: zero incidentes críticos; proteção contra OWASP Top 10; conformidade LGPD.

5. Conformidade (LGPD) e Segurança da Informação

- Bases legais e direitos do titular documentados (vide `frontend/public/legal/politica-privacidade-crowdfunding.md`).
- Tratamento mínimo de dados pessoais; uso de operadores (Clerk, Stripe, AWS S3, Vercel).
- Medidas técnicas: CSP, rate limit, idempotência, logs com redaction e anonimização de IP, armazenamento seguro de chaves/segredos.
- Dados de pagamento sensíveis não são armazenados pela aplicação; delegação ao Stripe (PCI-DSS pelo operador).

6. Observabilidade, Operação e Qualidade

- Logs estruturados com `X-Request-Id`; integração opcional com New Relic (`backend/newrelic.js`).
- Qualidade estática e cobertura: integração com Sonar e relatórios `coverage` / `coverage-unit`.
- Testes:
  - Backend: Jest (`backend/jest.config.js`), testes de controllers, services, utils e middleware.
  - Frontend: Vitest (`frontend/vitest.config.ts`), testes de utils e componentes.

7. Variáveis de Ambiente (mínimo)

- Backend:
  - `DATABASE_URL`
  - `CLERK_SECRET_KEY` ou `CLERK_JWT_VERIFICATION_KEY`
  - `STRIPE_SECRET_KEY`, `STRIPE_WEBHOOK_SECRET`
  - `FRONTEND_ORIGIN` (lista separada por vírgula), `APP_BASE_URL` / `CANONICAL_BASE_URL` (opcional), `ENFORCE_HTTPS=true|false`
  - `AWS_REGION`, `S3_BUCKET` (ou `AWS_S3_BUCKET`), `ASSETS_BASE_URL` (opcional para CDN/CloudFront)
- Frontend:
  - `VITE_CLERK_PUBLISHABLE_KEY`
  - `VITE_API_BASE_URL` (ex.: `http://localhost:3333` em dev)

8. Roadmap Resumido

- Iteração 1: Auth/CRUD básico de campanhas e listagem pública.
- Iteração 2: Stripe Checkout + Webhooks; comentários.
- Iteração 3: Imagens/Vídeos com S3 e UI.
- Iteração 4: SSE e métricas; Connect/assinaturas.
- Iteração 5: Observabilidade, polimento e melhorias de UX/A11y.

9. Referências

- Stripe API Documentation — `https://stripe.com/docs/api`

- Clerk Docs — `https://clerk.com/docs`
- OWASP ASVS & Top 10 — `https://owasp.org`
- LGPD (Lei nº 13.709/2018)
- REST API Design Guidelines (Microsoft)

— Fim —