Especificação Técnica para Desenvolvimento de Software

Instrução Detalhada de Arquitetura e Desenvolvimento

23 de agosto de 2025

1 Objetivo

Desenvolver uma aplicação web escalável (SPA + API) com autenticação segura, cobrança via Stripe, recursos de geolocalização, notificações (e-mail ou Telegram), documentação aberta via Swagger, testes (unitários e end-to-end), observabilidade e pipeline de CI/CD, toda empacotada em Docker.

2 Arquitetura

- **Frontend**: Next.js 15 (App Router), com *Tailwind + shadcn/ui*.
- **Backend**: NestJS, com Swagger e OpenAPI.
- **Banco de dados**: PostgreSQL + Prisma ORM.
- **Cache/Filas**: Redis + BullMQ.
- **Pagamentos**: Stripe (Checkout + Webhooks).
- **Geolocalização**: Mapbox (ou Google Maps/OSM).
- **Notificações**: SMTP/Resend ou Telegram.
- **Containerização**: Docker + Docker Compose.
- **CI/CD**: GitHub Actions.

3 Estrutura do Repositório

```
app/
  apps/
    web/       # Next.js (frontend)
    api/       # NestJS (backend)
    worker/    # Processador de filas
  packages/
    ui/        # Componentes compartilhados
    config/    # Configura  es (eslint, tsconfig)
    types/     # Tipos compartilhados
```

```
  docker-compose.yml
  .env.example
  .github/workflows/ci.yml
  README.md
```

4 Docker Compose (desenvolvimento)

```
version: "3.9"
services:
  postgres:
    image: postgres:16-alpine
    environment:
      POSTGRES_USER: app
      POSTGRES_PASSWORD: app
      POSTGRES_DB: appdb
    ports: ["5432:5432"]

  redis:
    image: redis:7-alpine
    ports: ["6379:6379"]

  api:
    build: ./apps/api
    env_file: .env
    depends_on: [postgres, redis]
    ports: ["3000:3000"]

  web:
    build: ./apps/web
    env_file: .env
    depends_on: [api]
    ports: ["3001:3001"]
```

5 Variáveis de Ambiente

```
DATABASE_URL=postgresql://app:app@postgres:5432/appdb
JWT_ACCESS_SECRET=changeme
STRIPE_SECRET_KEY=
EMAIL_PROVIDER=smtp
MAPBOX_TOKEN=
TELEGRAM_BOT_TOKEN=
REDIS_URL=redis://redis:6379
```

6 Ambientes

O sistema deverá contemplar três ambientes distintos:

6.1 Desenvolvimento (Development)

- Executado localmente via `docker-compose up`.

- Banco de dados e Redis com persistência local.
- Permitir recarga automática de código (hot reload).
- Uso de chaves de API em modo *sandbox* (Stripe, Mapbox, etc.).

6.2 Homologação (Staging)

- Ambiente intermediário para validação antes do deploy em produção.
- Banco de dados isolado, preferencialmente resetável a cada ciclo.
- Deploy automatizado via **branch staging** no GitHub.
- Testes de integração e E2E obrigatórios antes da liberação.
- Uso de chaves de API em modo *test*.

6.3 Produção (Production)

- Ambiente final, com foco em disponibilidade e segurança.
- Deploy automatizado via **branch main**.
- Banco de dados com backup periódico.
- Monitoramento (logs, métricas, alertas).
- Uso de chaves de API em modo *live*.
- TLS obrigatório com certificados válidos.

7 Passos Iniciais do Projeto

A seguir estão descritos os 16 passos iniciais necessários para o desenvolvimento do sistema, com explicações detalhadas de cada etapa.

1. **Bootstrap & Infraestrutura**
   Estruturar o repositório, preparar Dockerfiles e o `docker-compose.yml`, configurar TypeScript, Jest e pipeline de CI inicial.
2. **Autenticação e Usuários**
   Implementar autenticação com JWT, refresh tokens e RBAC. Criar módulo de usuários com cadastro e gerenciamento de perfis.
3. **Módulo de Pagamentos (Stripe)**
   Integrar Stripe no modo teste, com endpoints de checkout e webhooks. Simular transações para validar os fluxos de cobrança.
4. **Geolocalização**
   Conectar a provedores de mapas (Mapbox/Google/OSM) para autocomplete de endereços e renderização de mapas. Cachear resultados em Redis.

5. **Notificações**
   Criar serviço de envio via SMTP/Resend e fallback para Telegram. Usar templates para mensagens de confirmação e alertas.

6. **Documentação e Observabilidade**
   Gerar documentação automática da API com Swagger, implementar health checks, logs estruturados e métricas para monitoramento.

7. **Testes End-to-End e Hardening**
   Implementar testes E2E cobrindo login, pagamentos e notificações. Adotar práticas de segurança como CORS restrito, Helmet e rate limiting.

8. **Deploy Inicial**
   Configurar pipelines de CI/CD com deploy automatizado para staging e produção, incluindo migrations e build de imagens Docker.

9. **Critérios de Aceitação (DoD)**
   Definir padrões claros: stack sobe via `docker-compose`, Swagger cobre todas as rotas, Stripe em modo teste funciona e notificações são entregues.

10. **Documentação do Projeto**
    Criar README com instruções de uso e `docs` auxiliares: arquitetura, segurança e operações (backup, deploy, monitoramento).

11. **Feature Flags**
    Adicionar sistema de flags para habilitar/desabilitar recursos em produção sem novo deploy, permitindo testes A/B.

12. **Armazenamento de Arquivos (S3/Minio)**
    Configurar upload de arquivos em S3 ou Minio, com suporte a links temporários e exclusão de arquivos.

13. **Mecanismo de Busca (Opcional)**
    Preparar integração com Postgres Full-Text Search ou Meilisearch, caso o sistema precise de busca avançada.

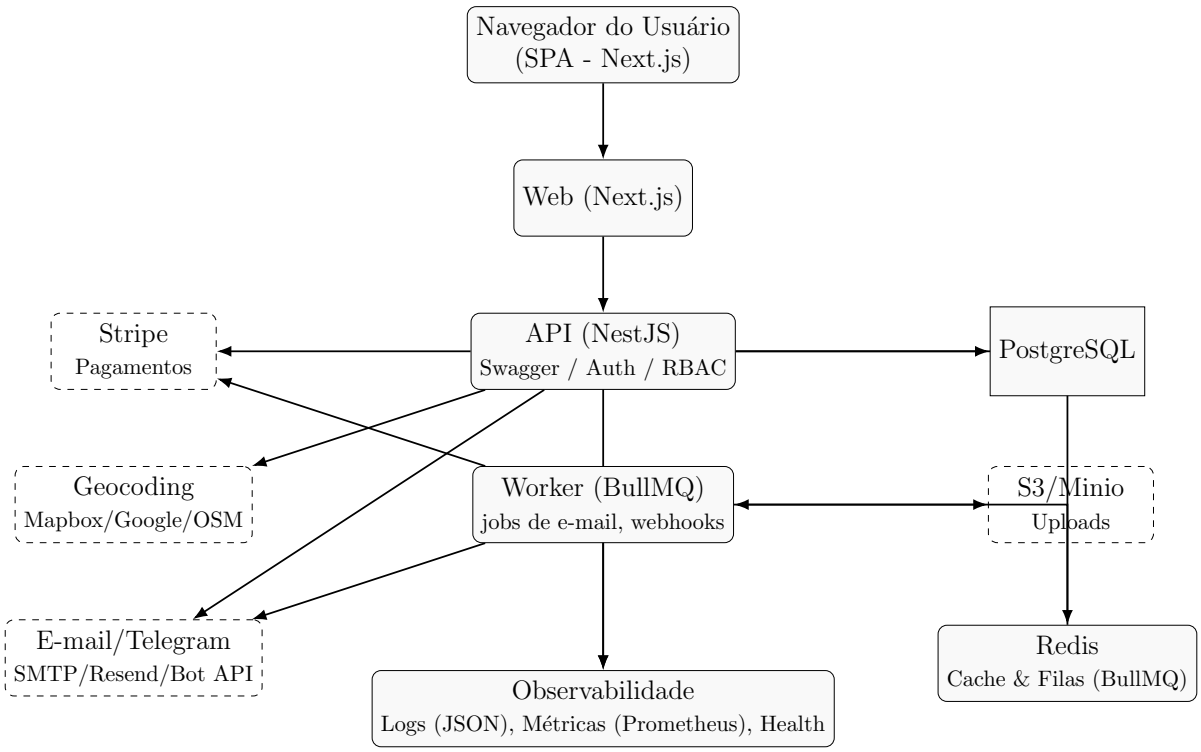
14. **Webhooks Outbound**
    Criar suporte a webhooks assinados (HMAC) para notificar sistemas externos de eventos do sistema.

15. **Admin UI (Painel de Administração)**
    Desenvolver interface administrativa com Next.js + shadcn/AntD para gerenciar usuários, permissões e logs.

16. **Service Mesh e API Gateway (Futuro)**
    Planejar adoção de ferramentas como Traefik/Kong para API Gateway e, futuramente, Istio como service mesh.

8 Diagramas

8.1 Arquitetura Lógica (Frontend, Backend e Serviços)



8.2 Pipeline de CI/CD e Ambientes (Dev → Staging → Prod)

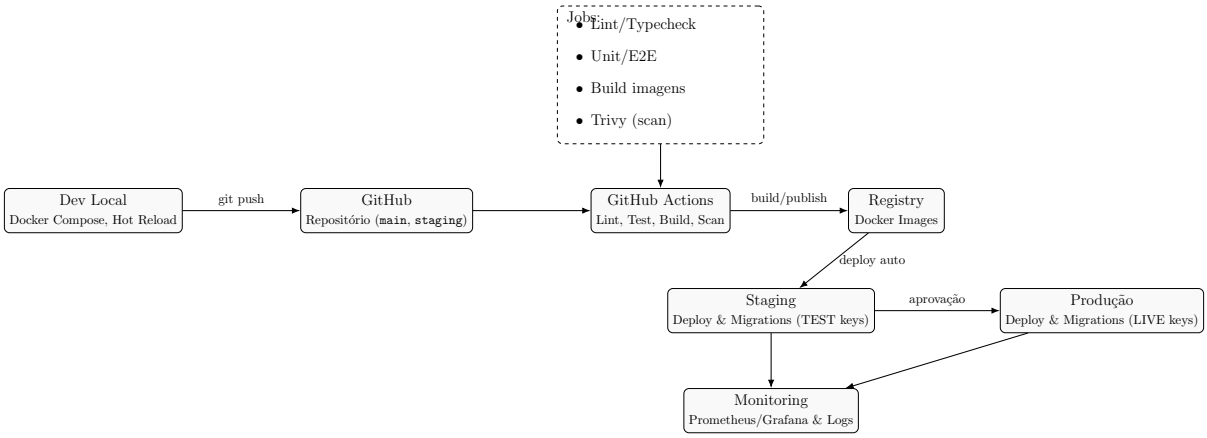


Figura 1: Fluxo de CI/CD e promoção entre ambientes.

9 Backend (NestJS)

- Módulos: Auth, Users, Billing (Stripe), Geo, Notify, Files, Health.
- Validação: class-validator + class-transformer.
- Documentação: Swagger em /docs.

- Notificações: abstração com implementação SMTP e Telegram.
- Segurança: Helmet, CORS, Rate Limiter, RBAC.
- ORM: Prisma (migrations e seed).

10 Frontend (Next.js)

- UI: shadcn/ui + Tailwind.
- Estado: React Query.
- Auth: next-auth ou JWT com cookies httpOnly.
- Mapas: Mapbox GL JS ou Google Maps.
- Testes: React Testing Library + Playwright.

11 Testes e Qualidade

- Unitários: Jest.
- End-to-end: Playwright.
- Lint/Format: ESLint + Prettier.
- Segurança: npm audit + Trivy (scan de imagens).

12 CI/CD

Pipeline com GitHub Actions incluindo:

1. Lint
2. Testes
3. Build
4. Scan de vulnerabilidades
5. Deploy automático

13 Perguntas de Customização

1. Usar **shadcn+Tailwind** como padrão de UI?
2. Provedor principal de geolocalização: Mapbox, Google ou OSM?
3. Notificações obrigatórias por e-mail ou via Telegram?
4. Usar Prisma ou TypeORM?
5. Incluir uploads S3/minio?
6. Incluir admin UI com RBAC?