Patri Tech — Whitepaper

Data: 2026-01-17
Autores: Matheus Costa (e equipe: Jackie, Cris, Evelyn)
Resumo: Versão condensada criada a partir do repositório matheus-costa-dev/PATRITECH.

1. Resumo executivo

Patri Tech é um sistema de gestão patrimonial e rastreio inteligente desenvolvido como projeto integrador. Seu objetivo é permitir o cadastro, controle, inventário e rastreamento de ativos físicos usando uma interface web moderna, autenticação segura e integração com um backend Supabase (PostgreSQL). O sistema inclui geração/uso de QR Codes, CRUD de ativos, gestão de lotes, e fluxo de redefinição de senha.

2. Problema a resolver

Organizações de pequeno e médio porte frequentemente:

- Perdem visibilidade sobre localização e estado de seus ativos;
- Enfrentam processos manuais para inventário e auditoria;
- Precisam de uma solução leve, acessível e rápida de implementar que integre autenticação e banco em nuvem.

Patri Tech propõe centralizar a gestão patrimonial, reduzir erros manuais e facilitar auditorias e verificações de ativos em campo via QR Code.

3. Proposta de solução

Patri Tech oferece:

- Cadastro centralizado de ativos, categorias, condições e localizações;
- Rastreamento por QR Code para identificar rapidamente cada ativo;
- Fluxo de autenticação via Supabase Auth com recuperação de senha;
- Interface web responsiva (Next.js + React + Tailwind) para operações de CRUD e relatórios;
- Backend e persistência com Supabase (Postgres), permitindo RLS e políticas de segurança.

Benefícios: implantação rápida (Vercel + Supabase), uso de tipos TypeScript para coerência e componentes reutilizáveis.

4. Arquitetura e componentes principais

- Frontend: Next.js (app router) em TypeScript + React. Layout global com Navbar, Footer, Toasts e provedor de autenticação.
- Estilos: Tailwind CSS.
- Backend / DB: Supabase (PostgreSQL). Supabase também fornece Auth.
- Comunicação: Cliente Supabase no frontend e server actions Next.js para operações que exigem execução no servidor.
- Componentes notáveis: ResultTable (tabelas), Hero (login), formulários de ativos/lotes.
- Observabilidade: páginas como `/ativos`, `/lotes`, e rota de recuperação de senha `/redefinir-senha`. Robots/sitemap gerados programaticamente.

Diagrama lógico (texto):
Frontend (Next.js) ↔ Supabase Client ↔ Supabase (Auth + Postgres) Componentes: AuthContext → Proteção de rotas → CRUD Ativos → Storage/QR (opcional)

5. Modelo de dados (resumo)

Exemplo conciso das tabelas sugeridas (ajustar conforme implementação final):

- usuarios (id uuid, nome, email, role, criado_em)
- categorias (id_categoria, nome_categoria)
- condicoes (id_condicao, nome_condicao, gera_avaria)
- localizacoes (id_localizacao, nome_localizacao)
- lotes (id_lote, nome_lote, descricao, criado_em)
- ativos (id_ativo uuid, nome_ativo, id_categoria, id_condicao, id_localizacao, id_usuario_criador, id_lote, data_ultima_verificacao, data_criacao)

Nota: confirmar nomes exatos de colunas no Supabase (o repositório indica inserções como `supabase.from('ativos').insert([{ Item: item }])` — atenção a case sensitivity e mapeamento de campos).

6. Segurança e boas práticas

- Autenticação: usar Supabase Auth e tratar tokens com cuidado.
- Row-Level Security (RLS): ativar em produção e definir policies para permitir apenas operações autorizadas (ex.: dono do ativo ou roles específicos).
- Segredos: manter chaves de serviço (SERVICE_ROLE) fora do client (não usar NEXT_PUBLIC_ para chaves privilegiadas).
- Validação/Tratamento de erros: conferir retornos `error` do Supabase e notificar o usuário apropriadamente.
- Migrations: versionar schema SQL (migrations) para replicabilidade.

7. Fluxos importantes

- Login: via AuthContext -> signInWithPassword (Supabase). Redirecionamento para rotas protegidas.
- Reset de senha: envio de email com redirectTo para `/redefinir-senha`.
- Inserção de ativos: formulários chamam server actions (`'use server'`) que gravam via cliente Supabase e revalidam rotas.
- Inventário: listagem paginada/filtrada em tabela (react-data-table-component), possibilitando export/relatórios.

8. Tecnologias e dependências

Stack principal:

- Next.js (app router), TypeScript, React
- Supabase (Auth + Postgres)
- Tailwind CSS
- Pacotes auxiliares: react-toastify, react-data-table-component, react-icons, next/font
- Hospedagem sugerida: Vercel (frontend) + Supabase (DB/API)

9. Implantação e requisitos

Ambiente (ex.: .env.local):

- NEXT_PUBLIC_SUPABASE_URL
- NEXT_PUBLIC_SUPABASE_ANON_KEY
- SUPABASE_SERVICE_ROLE_KEY (somente servidor)
- Variáveis adicionais para redirects e integrações

Passo a passo resumido:

1. Criar projeto no Supabase e as tabelas (executar migrations SQL sugeridas).
2. Ajustar RLS e policies.
3. Configurar variáveis no Vercel.
4. Deploy do frontend em Vercel.
5. Verificar URLs de redirect de reset de senha.

10. Roadmap e recomendações

Curto prazo:

- Validar nomes de colunas e consolidar migrations.
- Implementar geração/scan de QR Codes com vinculação clara ao campo `id_ativo`.
- Testes e tratamento de erros para server actions.

Médio prazo:

- Auditoria de acessos e logs de verificação de ativos.
- Módulo móvel (PWA ou app) para leitura de QR em campo.
- Integração com sistemas de inventário e relatórios exportáveis (CSV / PDF).

Longo prazo:

- Regras avançadas de workflow (reservas, transferências entre localizações).
- Analytics de ciclo de vida de ativos e previsão de manutenção.

11. Conclusão

Patri Tech fornece uma base moderna, escalável e rápida para gestão patrimonial, combinando uma interface Next.js com o backend gerenciado Supabase. O foco em boas práticas (RLS, migrations, TypeScript) facilita evolução e adoção por organizações que necessitam de controle de inventário e rastreabilidade via QR Code.

12. Contato

Projeto no GitHub: https://github.com/matheus-costa-dev/PATRITECH