Relatório de Estudo — PedagoPass

(gerado em 2025-11-08 15:55)

1) Visão Geral

PedagoPass é uma plataforma para organizar e facilitar experiências de viagem/estudo para professores e comunidades escolares. O projeto está dividido em **Frontend** (Next.js 14 + TypeScript + Tailwind) e **Backend** (Node.js + Express + Prisma + MySQL).

- **Frontend** (`/src`): rotas no padrão App Router (`/src/app`), componentes em `/src/components` , estilos em `/src/styles` .
- **Backend** (`/backend`): API REST em Express, autenticação JWT (Bearer por padrão), acesso a banco via Prisma Client.
- **Infra**: variáveis `.env` e guias de deploy (Railway/Netlify) já inclusos no repositório.

2) Principais Funcionalidades (o que o sistema faz e como faz)

A. Autenticação

- **Fluxo**: formulário de login/cadastro no Frontend → chamada à API (`POST /api/auth/login` e `POST /api/auth/signup`) → validação → retorno de **JWT (Bearer)** e dados do usuário.
- **Destaques**: modo alternativo de **Cookie HttpOnly** previsto no código; **Login Rápido** via *quick token* com expiração curta (`POST /api/auth/quick-token` + `POST /api/auth/login/quick`).

B. Comunidades

- **Listagem**: `GET /api/communities` traz comunidades com tags, contagem de membros etc.
- **Detalhe**: `GET /api/communities/:slug` retorna metadados de uma comunidade.
- **Entrar/Sair**: `POST /api/communities/:slug/join` e `DELETE /api/communities/:slug/join` (requerem autenticação).

C. Reservas (de viagens/pacotes)

- **Criar**: `POST /api/reservations` registra uma nova reserva (destino, datas, participantes, forma de pagamento). Status inicial: `pendente` .
- **Consultar**: `GET /api/reservations/me` e `GET /api/reservations/:id` .
- **Atualizar status**: `PATCH /api/reservations/:id/status` para `pendente | confirmada | cancelada` (validação de dono).

D. Pedidos/Ordens (pagamento)

- **Emitir cobrança / marcar pago**: `POST /api/orders/mark-paid` gera um `Order` a partir de uma `Reservation` e ajusta o status para `confirmada` .
- **Consulta**: `GET /api/orders/me` , `GET /api/orders/:id` , `GET /api/orders/by-reservation/:reservationId` .

E. “Minha conta”

- **Perfil básico**: `GET /api/me` e `GET /api/me/communities` (listar comunidades do usuário).
- **Alterar senha**: `POST /api/me/password` (com verificação de senha atual).

Observação para apresentação: foque na jornada “Cadastro → Login → Explorar Comunidades → Fazer uma Reserva → Marcar Pago”, que demonstra o *core* do produto em poucos minutos.

3) Arquitetura (alto nível)

- **Front**: Next.js 14 (App Router). UI em componentes reutilizáveis; dados acessados via `fetch` para a API (`NEXT_PUBLIC_API_URL`). Sessão simples baseada em cookie (`src/server/session.ts`) para cenários sem backend.
- **Back**: Express com middlewares (CORS, Helmet, Cookie-Parser). Rotas montadas em `/api` e `/api/auth`. Log e segurança básicos.
- **Banco**: MySQL (ou compatível) via Prisma. Entidades **User**, **Community**, **CommunityMembership**, **Reservation**, **Order**, **QuickToken**; `ReservationStatus` como enum.

4) Endpoints da API

A base aceita tanto `/{{endpoint}}` quanto `/api/{{endpoint}}` (ver `backend/src/index.ts`).

Tabela de Endpoints

(Abra o anexo "Endpoints da API (PedagoPass)" na lateral para a tabela interativa.)

5) Modelagem de Dados (Prisma)

A seguir, campos, tipos e constraints das principais entidades.

Community

Campo	Tipo	Atributos
`slug`	`String`	@id
`nome`	`String`	
`descricao`	`String?`	
`tags`	`String?`	@db.Text
`membros`	`Int`	@default(0)
`destinoSlug`	`String?`	
`destinoNome`	`String?`	
`destinoImagem`	`String?`	
`destinoCidade`	`String?`	

Campo	Tipo	Atributos
`capa`	`String?`	
`createdAt`	`DateTime`	@default(now())

CommunityMembership

Campo	Tipo	Atributos
`id`	`Int`	@id @default(autoincrement())
`userId`	`String`	
`slug`	`String`	
`createdAt`	`DateTime`	@default(now())
`user`	`User`	@relation(fields: [userId], references: [id], onDelete: Cascade)
`community`	`Community`	@relation(fields: [slug], references: [slug], onDelete: Cascade)

Constraints únicas - `@@unique([userId, slug])`

Order

Campo	Tipo	Atributos
`id`	`String`	@id @default(cuid())
`userId`	`String`	
`reservationId`	`String?`	
`destinoSlug`	`String`	
`destinoNome`	`String`	
`total`	`Float`	
`metodo`	`String`	
`parcelas`	`Int?`	
`pagoEm`	`DateTime?`	
`createdAt`	`DateTime`	@default(now())

Índices - `@@index([userId])` - `@@index([reservationId])`

QuickToken

Campo	Tipo	Atributos
`id`	`String`	@id @default(cuid())

Campo	Tipo	Atributos
`userId`	`String`	
`token`	`String`	@unique
`expiresAt`	`DateTime`	
`used`	`Boolean`	@default(false)
`createdAt`	`DateTime`	@default(now())
`user`	`User`	@relation(fields: [userId], references: [id], onDelete: Cascade)

Índices - `@@index([userId])` - `@@index([expiresAt])`

Reservation

Campo	Tipo	Atributos
`id`	`String`	@id @default(cuid())
`userId`	`String`	
`destinoSlug`	`String`	
`destinoNome`	`String`	
`destinoImagem`	`String?`	
`inicio`	`DateTime`	
`fim`	`DateTime`	
`participantes`	`Int`	@default(1)
`formaPagamento`	`String?`	
`totalEstimado`	`Float`	
`status`	`ReservationStatus`	@default(pendente)
`createdAt`	`DateTime`	@default(now())

User

Campo	Tipo	Atributos
`id`	`String`	@id @default(cuid())
`nome`	`String`	
`email`	`String`	@unique
`passwordHash`	`String`	
`createdAt`	`DateTime`	@default(now())

Enums

- **ReservationStatus**: `pendente`, `confirmada`, `cancelada`

Relações (resumo): - `User 1–N CommunityMembership` (membros em comunidades) - `User 1–N Reservation` (reservas criadas pelo usuário) - `User 1–N Order` (pedidos/ordens de pagamento) - `User 1–N QuickToken` (tokens de login rápido) - `Community 1–N CommunityMembership` - `Reservation 1–N Order` (cada reserva pode gerar ordens; no fluxo atual, envolve confirmação de pagamento)

6) Variáveis de Ambiente

- **Frontend** (`.env`):
- `NEXT_PUBLIC_API_URL` → URL base da API (ex.: `https://.../api`)
- **Backend** (`backend/.env`):
- `DATABASE_URL` → conexão MySQL (ex.: `mysql://user:pass@host:3306/db`)
- `JWT_SECRET` → segredo para assinar tokens JWT
- `CORS_ORIGIN` → origens permitidas separadas por vírgula (ex.: `https://seusite.com, http://localhost:3000`)
- `PORT` (opcional, padrão 8080)

Há guias prontos no repo: **SETUP_NETLIFY.md**, **RAILWAY_SETUP.md**, **PRODUCTION_CHECKLIST.md** e **FIX_CORS.md**.

7) Scripts úteis

Frontend (`package.json`): - `dev` → `next dev` - `build` → `next build` - `start` → `next start` - `env:check:frontend` → verificador de envs do front

Backend (`backend/package.json`): - `dev` → `tsx watch src/index.ts` - `build` → `tsc` (gera `dist/`) - `start` → `node dist/index.js` - `prisma:generate | :push | :deploy | :seed` → ciclo Prisma - `prestart` roda `prisma migrate deploy` automaticamente (útil em produção)

8) Fotos & Assets

As imagens de destinos/comunidades estão em `/public/images`. Exemplos (14 encontradas): belem-pa-2.webp, brasilia.jpg, esportes.jpg, inhotim-capa2019-01.jpg, manaus.jpg, olinda.jpg, ouro-preto-conheca-a-cidade-historica-mais-famosa-de-minas-gerais.jpg, paraty.jpg, recife.jpg, rj.jpg … - **Como trocar/adicionar**: salve o arquivo em `public/images` e referencie pelo caminho `/images/arquivo.ext`. - **Boas práticas**: 1600×900+ (16:9), *webp* preferível, nomes descritivos (ex.: `rio-corcovado.webp`).

9) Como explicar em apresentação (roteiro de 6–8 minutos)

1. **Problema** (30s): professores e escolas precisam organizar viagens educativas com mais segurança e clareza.
2. **Solução** (45s): plataforma que conecta comunidades, destinos e reservas com controle de pagamento.

3. **Arquitetura** (60s): front Next.js 14 + back Express/Prisma em MySQL. Mostre o diagrama das entidades e os endpoints principais.
4. **Demo guiada** (3–4 min): Cadastro → Login → Explorar Comunidades → Criar Reserva → Marcar Pago → Ver status.
5. **Segurança & dados** (45s): JWT, enum de status, constraints de unicidade, índices.
6. **Deploy** (30–60s): Netlify no front e Railway no back; `.env` simplificadas e `CORS_ORIGIN`.

Confira antes: variáveis `.env` preenchidas, backend online, imagens presentes, e script `env:report` / `env:check` sem erros.

10) Perguntas comuns (e respostas curtas)

- **Como o login funciona?** → Via **JWT Bearer** (padrão). Há opção de **Cookie HttpOnly**. Existe **login rápido** via *quick token* com expiração e marcação de uso.
- **E se a reserva for cancelada?** → `PATCH /api/reservations/:id/status` aceita `cancelada`; a *order* vinculada pode ser ajustada conforme regra de negócio.
- **Onde ficam as comunidades do usuário?** → `GET /api/me/communities` cruza `CommunityMembership` com `Community`.
- **Como garantir CORS certo?** → Configure `CORS_ORIGIN` com as origens do front (produção e dev).

11) Próximos passos recomendados

- **Validações adicionais** no back (campos obrigatórios e formatos).
- **Testes automatizados** (unitários no back e *e2e* no front).
- **Observabilidade** (Pino + logs estruturados; métricas/trace se necessário).
- **RBAC simples** (admin para curadoria de comunidades/destinos).

> **Anexo técnico:** As tabelas de endpoints (acima) e o detalhamento Prisma servem como "cola" técnica durante a apresentação e para estudo rápido.