Objetivo

- Garantir operação local contínua em cada estabelecimento mesmo com falhas globais de Cloudflare/Internet.
- Separar o sistema em dois domínios: Edge (Local) e Core (Central/SaaS) com sincronização assíncrona e tolerância a falhas.

Viabilidade

- Alta: a aplicação atual (SPA + API) pode ser particionada em serviços com banco local (SQLite/Postgres) e banco central (Postgres), adicionando um motor de sincronização.
- Sem dependência única: Cloudflared passa a ser apenas um dos meios de acesso externo; operação local ocorre via LAN e sincroniza depois.

Visão de Arquitetura

- Edge Node (Local): serviço backend + UI local rodando em um PC do estabelecimento; exposto na LAN (`http://pc-local:6001`).
- Core (Central/SaaS): portal da assinatura + gestão geral da empresa + APIs multi-tenant.
- Sync Engine: camada comum que publica/consome mudanças (outbox/inbox), resolve conflitos e garante idempotência.
- Comunicação: HTTPS/WebSocket ao Core com backoff + fila local; opcional malha privada (VPN/Tailscale) para casos inter-estabelecimento.

Separação de Responsabilidades

- Assinatura/Licenças (Core): emissão/renovação de licenças assinadas; revogação; billing.
- Controle Geral da Empresa (Core): usuários, perfis/cargos, estabelecimentos, relatórios, consolidação de dados, backups.
- Sistema Local (Edge): cadastro/atendimento/prescrição/estoque, agendamentos, baixa de materiais, emissão de documentos; cache de permissões; fila offline.

Persistência e Multitenancy

- Core: Postgres com `tenantId` (empresa) e `establishmentId` (filial), chaves globais, auditoria.
- Edge: SQLite/Postgres com subset de tabelas por estabelecimento; metadados `originNodeId`, `version`, `updatedAt`, `syncStatus`.
- Identidades: `companyId`, `establishmentId`, `nodeId` (instância local) e `userId`.

Sincronização e Conflitos

- Outbox Pattern no Edge: toda alteração grava evento em `outbox` e aplica localmente.
- Inbox no Core: recebe eventos, valida licenças/permissões, faz upsert idempotente.
- Estratégia de conflito: last-write-wins com `updatedAt` + domínios críticos com regras específicas (ex.: estoque usa compensações), logs de auditoria.
- Resync bidirecional: Core devolve atualizações relevantes ao Edge (perfis, catálogos, bloqueios).

Tolerância a Falhas

- Operação Offline: UI local funciona sem Internet; servidor local aceita usuários via LAN; fila persiste em disco.
- Grace Period de Licença: token assinado com validade; sem conexão mantém modo restrito até expirar.
- Reconnect: retry exponencial + compactação de lotes; verificação de integridade (checksums) após retomada.

Segurança

- Licenças com assinatura (RSA/ECDSA) verificáveis offline.
- Service Account por `nodeId` com escopos mínimos; rotação periódica.
- Criptografia em repouso (Edge/Core), TLS em trânsito, auditoria completa; sem segredos na UI.

Pastas (Monorepo)

- `core/` — API SaaS (auth, licenças, empresa, relatórios) e Portal Web.
- `edge/` — serviço local (API + UI local) com banco local e fila offline.
- `packages/shared/` — tipos, esquemas Zod, modelos Prisma comuns.
- `packages/sync-engine/` — outbox/inbox, transporte, resoluções de conflito.
- `packages/auth/` — JWT, verificação de licença, permissões.
- `infra/` — Dockerfiles, compose para Edge, manifests de Core.

Fluxos

- Login Local: usuário se autentica no Edge; permissões cacheadas do Core; validação offline dentro do período.
- Sincronização: Edge publica eventos; Core consome; Core emite atualizações; Edge aplica.
- Inter-Estabelecimento: via Core quando online; offline cada Edge opera isolado e reconcilia depois.

Deploy

- Edge: `docker-compose` com `edge-api`, `edge-web`, `db` (SQLite/Postgres); exposto em LAN, sem cloudflared obrigatório.
- Core: Kubernetes/VM com `core-api`, `core-web`, Postgres, Redis opcional; observabilidade.

Roadmap de Implementação

1. Criar `packages/shared` e `packages/auth` para unificar tipos/esquemas/permissões.
2. Extrair módulos críticos do backend atual para `core/` (auth, licenças, empresa) e `edge/` (atendimento/estoque).
3. Implementar outbox no Edge e ingestão idempotente no Core; definir eventos por entidade.
4. Introduzir verificação de licença offline (assinaturas) e grace period.
5. Ajustar UI: portal Core (gestão/relatórios) e UI local (operações). Reuso máximo de componentes.
6. Criar Docker Compose para Edge e pipeline de deploy para Core.
7. Testar falha de Internet/Cloudflare: operação local, re-sync, resolução de conflitos.
8. Observabilidade e auditoria: logs, métricas, trilhas por evento.

Critérios de Aceite

- Edge funcional sem Internet por 24–72h com dados persistidos.
- Re-sync completo sem perdas com validação de integridade.
- Licenças e permissões aplicadas corretamente em modo offline.
- Inter-estabelecimento funcionando quando online via Core.

Plano de Reorganização Atual → Final

- Mapear pastas atuais (frontend/backend) e migrar para `core/` e `edge/` mantendo padrões.
- Centralizar schemas e validações em `packages/shared` para evitar divergências.
- Isolar transporte/sincronização em `packages/sync-engine` para reutilização.

Riscos e Mitigações

- Conflitos de dados: definir regras por domínio, usar idempotência e auditoria.
- Divergência de schema: CI para validar shared contra Edge/Core.
- Segurança de chaves: vault e rotação, tokens de curta duração.

Próximo Passo

- Confirmar este plano e iniciar a fase 1 (shared/auth + outbox básico no Edge).