README - Roadmap Blockchain para Histórico Médico (Foco: Camada Blockchain)

Visão Geral

Este projeto tem foco na **camada blockchain** de um sistema de Histórico Médico onde:

- **O paciente é proprietário dos seus dados**;
- Registros médicos (hashes + metadata) e **senhas sequenciais de atendimento** (tickets com data/hora) são mantidos on-chain;
- Dados sensíveis (laudos, imagens, PDFs) são mantidos **off-chain** (S3/IPFS/MinIO) e referenciados por hash/URI;
- A solução é **permissionada** (rede consórcio) com governança e autenticação de entidades (hospitais, clínicas, laboratórios — denominadas "agências") e pacientes;

> Observação: este README foca exclusivamente na arquitetura blockchain, modelos on-chain, consensos, governança, segurança e regras de negócio. Front-end e apps são considerados melhorias futuras.

Objetivos Principais

1. Registrar metadados de **registros médicos** (id, patient_did, record_hash, storage_uri, creator, timestamp);
2. Permitir que o **paciente controle consentimentos** de leitura/escrita;
3. Registrar **senhas sequenciais de atendimento** (tickets) que contenham data e hora do atendimento e fiquem salvas no histórico do paciente e no histórico da agência (clínica/hospital);
4. Garantir **auditoria imutável** (quem fez o quê e quando);
5. Ser compatível com padrões de saúde (ex.: FHIR) para interoperabilidade futura.

Escolha da Plataforma (recomendação)

Principal recomendação: Substrate / FRAME (Rust) — por ser modular, permitir pallets customizados, e fácil integração com Polkadot-JS Apps para testes.
Alternativa forte: Hyperledger Fabric — excelente para consórcios empresariais, fácil controle de identidade (Fabric CA) e políticas de privacidade por canal.

Modelo de Dados On-Chain (Resumo)

Record (registro médico - metadata)

- `record_id: Hash` (UUID ou hash)
- `patient_did: DID` (identificador pseudonimizado do paciente)
- `record_hash: Bytes` (SHA-256 do blob cifrado)
- `storage_uri: Bytes` (IPFS CID ou S3 URI encriptado)

- `creator: AccountId` (quem inseriu o registro)
- `created_at: BlockNumber or Timestamp`
- `record_type: Enum` (ex: LAB, PRESCRIPTION, DIAGNOSIS)
- `version: u32` (versionamento)

Ticket (Senha sequencial de atendimento)

- `ticket_id: Hash`
- `agency_id: AccountId` (clínica/hospital que emitiu)
- `patient_did: DID`
- `sequence_number: u64` (sequencial por agência por dia)
- `issued_at: Timestamp`
- `appointment_time: Option<Timestamp>` (se aplicável)
- `status: Enum` (ISSUED, CALLED, COMPLETED, CANCELLED)
- `reference_record_id: Option<RecordId>` (opcional)

AccessGrant

- `record_id`
- `grantee: AccountId`
- `granted_at: BlockNumber`
- `expires_at: Option<BlockNumber)`

Audit Log

- Evento on-chain (RecordCreated, AccessGranted, TicketIssued, TicketStatusChanged, etc.)

Regras de Negócio (Básicas)

1. **Ownership**: paciente (por seu DID) é o controlador lógico do seu histórico. Registros podem ser criados por provedores (creator), mas o paciente controla concessões de leitura.
2. **Criação de registro**:
   - Apenas contas com papel `Provider` (hospital, laboratório) podem submeter `create_record` extrinsic.
   - O extrinsic grava `record_hash`, `storage_uri` e emite `RecordCreated`.
3. **Consentimento**:
   - `grant_access` e `revoke_access` são operações on-chain; apenas o paciente (ou um delegate autorizado) pode conceder/revogar acesso.
   - Quando `grant_access` é chamado, o backend off-chain (API) deverá re-encriptar a data key para o grantee via KMS/HSM.
4. **Tickets (senhas sequenciais)**:
   - Cada `agency` mantém um contador sequencial por **dia** (reset às 00:00 local da agência).
   - Formato da senha: `{agency_code}-{YYYYMMDD}-{seq:04d}-{HHMM}`.
     Ex.: `AG12-20251124-0007-0932` (agência AG12, data 2025-11-24, sequência 7, emitida 09:32).
   - Ao emitir um ticket, grava-se on-chain `TicketIssued` (com sequence_number e issued_at).
   - Tickets podem ser vinculados a um `record_id` posteriormente (ex: quando exame é requisitado).

5. **Privacidade**:
   - **NUNCA** salvar PII/PHI em texto aberto na chain.
   - Use pseudonímia (DID derivado com salt) e armazene somente hashes e URIs.
6. **Revogação**:
   - Revogação de acesso impede futuras concessões de data keys; não remove históricos já lidos.
7. **Auditoria**:
   - Todo acesso necessita gravação de evento `AccessLogged` (quem, quando, motivo, tx_id).
8. **Governança**:
   - Validações: nó validador com papel `Validator` fornecido por entidade do consórcio.
   - Políticas de on-boarding/off-boarding definidas off-chain e aplicadas por multisig governance (on-chain proposals optional).

Extrinsics / Smart Contract Interface (pallet API)

(Sintaxe inspirada em Substrate FRAME; adaptar conforme platform)

- `create_record(record_id, patient_did, record_hash, storage_uri, record_type)`
  Caller: Provider; *Effect:* grava `Record`, emite `RecordCreated`.
- `grant_access(record_id, grantee_account, expires_at)`
  Caller: Patient (or delegate); *Effect:* atualiza `AccessList`, emite `AccessGranted`.
- `revoke_access(record_id, grantee_account)`
  Caller: Patient; *Effect:* remove grant, emite `AccessRevoked`.
- `log_access(record_id, accessor_account, reason)`
  Caller: Provider backend after verifying access; *Effect:* emite `AccessLogged` event.
- `issue_ticket(agency_id, patient_did, appointment_time?) -> Ticket`
  Caller: Agency node/operator; *Effect:* increments sequence (per day), grava `Ticket` e emite `TicketIssued`.
- `change_ticket_status(ticket_id, new_status)`
  Caller: Agency; *Effect:* emite `TicketStatusChanged`.
- `get_patient_records(patient_did)` (RPC/Query)
  Caller: any (authorization handled off-chain or via grants).
- `get_agency_tickets(agency_id, date)` (RPC/Query)

Sequência de Emissão de Tickets (detalhes técnicos)

- **Atomicidade**: emissão deve ser an atomic extrinsic that increments the per-agency daily counter and returns the sequence.
- **Storage**: `DailyCounter<AgencyId, Date> -> u64`
- **Algorithm**:
  1. Calculate `date_key = formatYYYYMMDD(local_time_of_agency)`.

2. Load `seq = DailyCounter.get(date_key).unwrap_or(0) + 1`.
3. Write `DailyCounter[date_key] = seq`.
4. Compose `ticket_id = hash(agency_id || date_key || seq || timestamp)`.
5. Store `Ticket` map and emit `TicketIssued(ticket_id, agency_id, patient_did, seq, issued_at)`.

- **Concurrency**: as Extrinsics são sequenciais por bloco; conflitos são resolvidos naturalmente. Para alta concorrência, ajustar block time e weight.

Consenso & Deploy (rede permissionada)

- **Consensus**: PoA (Authority-based) ou Tendermint (BFT) dependendo da plataforma.
- **Node Types**:
  - Validators/Authorities (operam consenso)
  - Full nodes (fornecem APIs)
  - Archive nodes (para auditoria, long-term storage)
- **On-boarding**: cada agência única no consórcio passa por processo KYC e recebe uma `AccountId`/certificado.
- **Chaveamento**: validators usam chaves X25519/ed25519; rotacionamento de chaves suportado via governance.

Off-chain Components (visão mínima necessária)

Mesmo ignorando front-end, há componentes off-chain essenciais:

- **API Gateway / Provider Backend**:
  - Faz upload dos blobs cifrados ao storage;
  - Gera record_hash; chama extrinsic `create_record`;
  - Faz key wrapping / data key distribution quando `grant_access` é efetivado.
- **KMS/HSM**:
  - Armazena keys mestres; realiza envelope encryption.
- **Off-chain Workers / Relayers (Substrate)**:
  - Podem executar tarefas como pinning IPFS, monitorar expirations, automatizar key rotation.

Segurança & Compliance (essenciais)

- **Criptografia**:
  - Dados: AES-GCM 256 para blobs off-chain.
  - Keys: enveloping com RSA/ECIES ou X25519; data key cifrada para cada grantee.
- **Identidade**:
  - DID padrão (ex: DID:key, DID:web) com verifiable credentials para provar papéis (Provider, Agency).
- **Logging/Auditing**:
  - Eventos on-chain para todas as operações críticas.
- **LGPD/HIPAA**:

    - Planejar processos para requests de acesso/eliminação (dados pessoais permanecem off-chain).
- **PenTest & Privacy Impact Assessment** antes de produção.

Milestones (Roadmap focado em Blockchain) — 12 semanas

Milestone 0 — Preparação (1 semana)

- Escolher plataforma (Substrate recomendado) e template.
- Montar ambiente Rust/Substrate.
- Definir modelos de dados on-chain.
- Criar repositório e CI básico.

Entregáveis: Documento de design, `README`, CI pipeline.

Milestone 1 — Pallet Básico (2 semanas)

- Implementar pallet `medical_records` com:
    - Storage `Records`, `PatientRecords`.
    - Extrinsic `create_record`.
    - Event `RecordCreated`.
- Unit tests básicos.

Entregáveis: Pallet compilável com testes.

Milestone 2 — Access Control & Audit (2 semanas)

- Add `AccessList`, `grant_access`, `revoke_access`, `log_access` events.
- Implementar mapping DID↔Account (simplificado).
- Tests de autorização.

Entregáveis: Pallet com controle de acesso e testes.

Milestone 3 — Tickets (Senha Sequencial) (1.5 semanas)

- Implementar `Ticket` storage, daily counters e `issue_ticket`, `change_ticket_status`.
- Emissão on-chain com sequence format explained earlier.
- Tests de concorrência (multinode dev).

Entregáveis: Ticket system funcionando em dev node.

Milestone 4 — Integration Points (2 semanas)

- Off-chain worker / extrinsic hooks para:
    - Notificar backends quando `RecordCreated` ou `TicketIssued`.
    - Pinning IPFS (opcional).
- Implementar KMS integration stubs.

Entregáveis: Off-chain worker prototype; API stubs.

Milestone 5 — Multi-node & Consensus (2 semanas)

- Integrar pallet ao node-template.
- Levantar rede dev com 3 validators (PoA).
- Testes end-to-end: provider creates record, patient grants access, agency issues ticket.

Entregáveis: Devnet com 3 nodes, script de bootstrap.

Milestone 6 — Security, Docs & Handover (1.5 semanas)

- Security review (checklist), automated tests, documentation.
- Documentar processos de on-boarding e governance.
- Final README e steps to run.

Entregáveis: Security checklist, deployment guide, final README.

Tests & QA

- Unit tests do pallet (`#[cfg(test)]`).
- Integration tests (cargo test on runtime).
- Simulação multi-node (dev scripts).
- Fuzz testing for edge cases (ticket overflow, replay).
- Manual audit and pentest.

Operação & Monitoramento

- Monitorar: block time, TPS, extrinsic failures, event rates.
- Stack recomendada: Prometheus + Grafana, logs com ELK.
- Backup do storage off-chain (S3 lifecycle + versioning).
- Key rotation procedure documentado.

Entregáveis de Longo Prazo (após blockchain core)

- Frontend Patient Portal & Provider Portal.
- Search/index service (index hashes for fast lookup without exposing PII).
- Analytics privacy-preserving (differential privacy, aggregated metrics).
- Bridges / Interoperability com EHR via FHIR connectors.

Anexos Técnicos (Exemplo de formato de evento)

- `RecordCreated(record_id, patient_did_hash, creator, timestamp)`
- `AccessGranted(record_id, grantee, expires_at)`
- `TicketIssued(ticket_id, agency_id, patient_did_hash, seq_number, issued_at)`
- `TicketStatusChanged(ticket_id, old_status, new_status, changed_by)`

Observações Finais

Este roadmap é focado em construir uma camada blockchain sólida, auditável e privada para histórico médico com suporte nativo a **senhas sequenciais de atendimento** gravadas on-chain. A recomendação é iniciar com Substrate para maior flexibilidade de pallets e, numa segunda fase, integrar com infra de keys, off-chain storage e frontends.

Como abrir / rodar (dev quickstart)

1. Clonar repo (a ser criado) com node-template + pallet.
2. `rustup target add wasm32-unknown-unknown`
3. `cargo build --release`
4. `cargo run -- --dev`
5. Usar Polkadot-JS Apps para enviar extrinsics `create_record`, `grant_access`, `issue_ticket`.
