Decentralized Health Records System — Complete Project

React + DRF + Hyperledger Sawtooth (Python) — Full documentation, requirements, and example code

Generated: Full project package containing requirements, architecture, sample code snippets, Docker configuration, and instructions to run a local prototype. The PDF includes representative code for Sawtooth Transaction Processor (Python), DRF snippets, React scaffold snippets, Docker Compose, and deployment notes.

Table of Contents

1. Overview

This document describes a complete prototype blueprint for a Decentralized Health Records System (DHRS), where: - Patients own encrypted records stored off-chain (S3/MinIO) and anchor record hashes and consents on a Python-based blockchain (Hyperledger Sawtooth). - Doctors request access; patients grant/revoke consent on-chain. - DRF backend manages uploads, encryption, key wrapping, and interacts with Sawtooth via its REST API. - React frontend provides Patient Portal and Doctor Dashboard. - Optional Federated AI enables model training across hospital sites without sharing raw data.

2. Architecture & Components

Components: - React Frontend (walletless or certificate-based login): Upload, consent management, audit viewer. - Django REST Framework (DRF): APIs for user management, record uploads, consent operations, sawtooth client wrappers, and serving encrypted files. - Hyperledger Sawtooth Transaction Processor (Python): Handles REGISTER_PATIENT, REGISTER_DOCTOR, ADD_RECORD_HASH, GRANT_ACCESS, REVOKE_ACCESS, LOG_ACCESS. - Storage: Encrypted files in S3/MinIO; wrapped keys stored in Vault or KMS. - Federated AI: Flower server and site clients for training; model provenance anchored on-chain.

3. Tech Stack & Requirements

Recommended stack: - Frontend: React, Ethers.js/WebCrypto (for client-side decryption if used), Recharts. - Backend: Python 3.10+, Django 4.x, Django REST Framework. - Blockchain: Hyperledger Sawtooth (validator, REST API), Transaction Processor in Python. - Storage: MinIO or AWS S3, HashiCorp Vault or AWS KMS. - AI: Python, PyTorch or TensorFlow, Flower for federated learning. - DevOps: Docker, docker-compose, Postgres, Redis, Celery.

System Requirements (example):

Component	Minimum
CPU	4 cores
Memory	8 GB
Disk	50 GB
Python	3.10+
Docker	20.10+

4. Data Model (ERD gist)

Key entities (off-chain DB): - User (id, role [patient/doctor/admin], wallet_or_pubkey, created_at) - Record (id, patient_id, s3_path, hash_sha256, mime, created_at) - Consent (id, patient_id, doctor_id, scopes_json, key_ref, status, expiry) - AccessLog (id, record_id, doctor_id, tx_hash, timestamp) - Strategy/ModelProvenance (for AI) (id, model_hash, sites[], metrics) On-chain: only record anchors, consent entries (references to key_ref), and audit logs (no PHI).

5. Security & Compliance Highlights

- NEVER store plaintext PHI on-chain or in DB. - Encrypt files with AES-GCM; rotate patient data keys (PDK) on revoke. - Wrap PDK with recipient public key and store references in Vault/KMS. - Require doctor signature for LOG_ACCESS transactions so audit log reflects authorized access. - Implement RBAC in DRF and strong authentication (2FA optional). - Keep audit trails immutable via Sawtooth; provide tx links for verifiability. - Map flows to HIPAA/GDPR; implement data subject rights via off-chain workflows.

6. Setup & Run Instructions (Local Demo)

This demo assumes Docker & docker-compose installed. 1) Clone repo (or copy files provided in appendix). 2) Start Sawtooth validator & TP: - docker-compose -f docker/sawtooth/docker-compose.yaml up -d - Start your Python Transaction Processor (TP) in container or locally. 3) Start DRF backend: -

python -m venv .venv; source .venv/bin/activate - pip install -r backend/requirements.txt - python manage.py migrate - python manage.py runserver 4) Start frontend: - cd frontend; npm install; npm start 5) Use Postman or React UI to register patient, upload encrypted record, and anchor hash on-chain. Notes: - For local S3 emulation, use MinIO. - For key storage, HashiCorp Vault can be run via Docker for dev.

7. Code Samples

Below are representative, runnable snippets. For a full repo, see the appendix files.

7.1 Sawtooth Transaction Processor (Python) — handler.py

```
# blockchain/tx_processor/handler.py
from sawtooth_sdk.processor.core import TransactionProcessor
from sawtooth_sdk.processor.handler import TransactionHandler
from sawtooth_sdk.processor.exceptions import InvalidTransaction
import hashlib, cbor2

FAMILY_NAME = "ehr"
FAMILY_VERSION = "1.0"
def _ns(): return hashlib.sha512(FAMILY_NAME.encode()).hexdigest()[:6]
def make_address(key: str) -> str:
    return _ns() + hashlib.sha512(key.encode()).hexdigest()[:64]

class EHRHandler(TransactionHandler):
    @property
    def family_name(self): return FAMILY_NAME
    @property
    def versions(self): return [FAMILY_VERSION]
    @property
    def namespaces(self): return [_ns()]

    def apply(self, txn, context):
        payload = cbor2.loads(txn.payload)
        action = payload.get("action")
        signer = txn.header.signer_public_key

        if action == "REGISTER_PATIENT":
            pid = payload["patient_id"]
            addr = make_address(f"patient:{pid}")
            state_entries = context.get_state([addr])
            if state_entries and state_entries[0].data:
                raise InvalidTransaction("Patient exists")
            state = {"patient_id": pid, "owner": signer}
            context.set_state({addr: cbor2.dumps(state)})
            return

        # ADD_RECORD_HASH, GRANT_ACCESS, REVOKE_ACCESS, LOG_ACCESS similar...
```

7.2 DRF — Sawtooth client to submit batches (backend/apps/blockchain/client.py)

```
# backend/apps/blockchain/client.py (simplified)
import cbor2, hashlib, requests
from sawtooth_signing import create_context, CryptoFactory
from sawtooth_sdk.protobuf import transaction_pb2, batch_pb2

FAMILY_NAME = "ehr"; FAMILY_VERSION="1.0"
def make_batch(signer_key_hex, payload: dict):
    context = create_context('secp256k1')
    private_key = context.from_hex(signer_key_hex)
    signer = CryptoFactory(context).new_signer(private_key)
    payload_bytes = cbor2.dumps(payload)
    # build txn header and sign...
    # build batch and submit to Sawtooth REST API /batches
    return True
```

7.3 DRF — File upload, AES-GCM encryption, anchor hash

```
# backend/apps/records/utils.py (simplified)
import os, hashlib, base64
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

def encrypt_bytes(data: bytes):
    key = get_random_bytes(32)   # PDK
    iv = get_random_bytes(12)
    cipher = AES.new(key, AES.MODE_GCM, nonce=iv)
    ct, tag = cipher.encrypt_and_digest(data)
    return {"ciphertext": ct, "tag": tag, "iv": iv, "key": key}

def sha256_hex(data: bytes):
    return hashlib.sha256(data).hexdigest()
```

7.4 React — Consent Manager (simplified)

```
// frontend/src/components/ConsentManager.jsx (simplified)
import React, {useState, useEffect} from 'react';
import api from '../api';

export default function ConsentManager({patientId}){
  const [consents, setConsents] = useState([]);
  useEffect(()=>{ api.get(`/consent/?patient=${patientId}`).then(r=>setConsents(r.data)) },[patientId]);
  return (
    <div>
      <h3>Consents</h3>
      {consents.map(c=>(
        <div key={c.id}>
          <p>Doctor: {c.doctor_id} | Status: {c.status}</p>
          <button onClick={()=>api.post(`/consent/${c.id}/revoke/`)}>Revoke</button>
        </div>
      ))}
    </div>
  );
}
```

7.5 Docker Compose (sawtooth + minio + postgres + backend)

```
# docker-compose.yml (excerpt)
version: '3.7'
services:
  sawtooth-validator:
    image: hyperledger/sawtooth-validator:latest
    ports: ["4004:4004"]
  sawtooth-rest-api:
    image: hyperledger/sawtooth-rest-api:latest
    ports: ["8008:8008"]
  minio:
    image: minio/minio
    environment:
      MINIO_ROOT_USER: minio
      MINIO_ROOT_PASSWORD: minio123
    command: server /data
    ports: ["9000:9000"]
  backend:
    build: ./backend
    depends_on: ["sawtooth-rest-api","postgres"]
    ports: ["8000:8000"]
```

8. AI Federated Learning (Overview)

Use Flower (https://flower.dev) for federated training: - Each hospital runs a client that trains locally on private records (after proper preprocessing). - The central Flower server aggregates model updates. - After global aggregation, publish model provenance on-chain: - Submit ModelProvenance tx with model_hash, sites, metrics. Privacy: ensure differential privacy or secure aggregation if required by regulation.

9. Testing & Monitoring

Testing: - Unit tests for DRF (pytest/django). - Integration tests for Sawtooth transactions (use Sawtooth dev network). - E2E tests for React (Cypress). Monitoring: - Logs: ELK or Grafana Loki. - Metrics: Prometheus + Grafana. - Sentry for exception tracking.

10. Roadmap & Next Steps

M0: Baseline — Sawtooth TP + DRF anchor flow + React upload UI. M1: Consent key-wrapping workflow + doctor decrypt flow. M2: Access logs & audit viewer + clinical workflows. M3: Federated AI integration + model provenance on-chain. M4: Hardening (Vault/HSM), compliance audit, multi-node deployment. M5:

Optional: bridge to public chains for public verifiability of meta-proofs.

11. Appendix: Useful Commands & References

Useful commands: - docker-compose up -d - python manage.py migrate - python manage.py createsuperuser - sawset genesis (see Sawtooth docs) References: - Hyperledger Sawtooth docs: https://sawtooth.hyperledger.org - Flower (federated learning): https://flower.dev - Django REST Framework: https://www.django-rest-framework.org

End of document — prototype package. For a runnable repository and full source files, request 'generate repo' and I will produce a zip with code and a runnable docker-compose.