This Software Design Document (SDD) provides the technical blueprint for **Inkless**, focusing on the decentralized architecture, NIMC integration, and the local-first "Zero-Knowledge" flow.

## 1. Smart Contract Architecture (The Inkless Ledger)

The ledger serves as the source of truth for "Proofs." It does not store documents but anchors their cryptographic fingerprints.

### Contract: InklessRegistry.sol

* **Purpose:** Stores document hashes, manages signer permissions, and provides an immutable audit trail.
* **Logic:** \* **Anchor Hash:** Maps a docHash to a SignatureRecord.
  + **Access Control:** Only DIDs verified via NIMC can anchor signatures.
  + **Non-Repudiation:** Includes a hardwareID and vNINHash to bind the signature to a specific user and device.

### Data Structure: SignatureRecord

Solidity

struct SignatureRecord {  
 bytes32 docHash; // SHA-3 hash of the document  
 address signerDID; // The Decentralized ID of the signer  
 bytes pqcSignature; // Crystals-Dilithium signature  
 uint256 timestamp; // Block time of anchoring  
 bytes32 hardwareID; // Hash of the device TPM/Secure Enclave ID  
 bool isRevoked; // For compliance-based revocation  
}

## 2. Data Models & Database Schema

Inkless uses a **Relational Metadata Database** (stored in Nigeria) to handle user sessions and coordination, while the **Ledger** handles the proofs.

| **Table** | **Fields** | **Relationships** |
| --- | --- | --- |
| **Users** | id (UUID), did\_address, vnin\_hash, device\_pub\_key, created\_at | 1:M with AuditLogs |
| **AuditLogs** | id, user\_id, action\_type, ip\_address, timestamp | Linked to Users |
| **SignatureMetadata** | doc\_id (UUID), doc\_hash, signer\_id, ledger\_tx\_hash, status | 1:1 with Ledger Record |
| **VerificationTokens** | token\_id, doc\_hash, expiry, verification\_url | For sharing "Proof" |

## 3. API Endpoints

All API traffic must be routed through Nigerian-hosted gateways to comply with NITDA/NDPA.

### A. Identity (NIMC vNIN)

* **POST /api/v1/identity/verify**
  + **Request:** { "vNIN": "string", "consent\_token": "string" }
  + **Response:** { "status": "verified", "did": "did:inkless:0x...", "user\_profile": {...} }

### B. Signature Anchoring

* **POST /api/v1/signatures/anchor**
  + **Request:** { "docHash": "bytes32", "pqcSignature": "bytes", "hardwareID": "string" }
  + **Response:** { "txHash": "0x...", "anchoredAt": "ISO-Timestamp" }

### C. Verification (Public)

* **GET /api/v1/verify/{docHash}**
  + **Response:** { "isValid": true, "signer": "DID", "timestamp": "...", "ledgerTx": "0x..." }

## 4. Frontend Expectations & User Flow

The Frontend is a **Client-Side Heavy** application. It must handle cryptographic operations using **WebAssembly (Wasm)** so the document never touches the server.

### User Flow:

1. **Dashboard:** Renders a list of *Signed Metadata* (not the documents).
2. **Signing Step (Offline Capable):**
   * User drags PDF into the DropZone.
   * **Front-end Action:** Calculates SHA-3 locally.
   * **UI Display:** "Unique Fingerprint: 0x89...42. Your file is secure and has not been uploaded."
   * **Biometric Trigger:** App calls navigator.credentials.get() (WebAuthn) to sign the hash using the phone’s Secure Enclave.
3. **Synchronization:**
   * If **Online:** Send Signed Hash to POST /anchor.
   * If **Offline:** Store Signature locally in IndexedDB. Show a "Pending Sync" badge.

## 5. Developer Implementation Checklist

* [ ] **Crypto Module:** Implement dilithium-js or similar PQC library via WebAssembly.
* [ ] **Local Storage:** Use IndexedDB for storing signed hashes during offline sessions.
* [ ] **NIMC Sandbox:** Request access to the **NINAuth** API for vNIN validation.
* [ ] **Ledger Node:** Deploy a private EVM-compatible node on a Nigerian Cloud provider (e.g., MainOne/Rack Centre).

**Would you like me to generate the package.json with the required cryptographic libraries or provide a sample Solidity file for the InklessRegistry contract?**