Overview:
ChainDocs is a React + Vite SPA that uses Solana (via wallet-adapter and web3.js) to let institutions issue verifiable documents and holders discover, import, claim (mint-as-NFT via simulated flow) and verify them. The app uses on-chain Memo transactions for lightweight proof and localStorage as a UX cache/index.

High-level architecture:

- **Frontend:** React app (`App.jsx`, `main.jsx`) with routes for `Home`, `IssuerDashboard`, `HolderDashboard`, `VerifyDocument`, `WalletGuide`.
- **Wallet & RPC:** Solana wallet adapters (`@solana/wallet-adapter-*`) are configured in `App.jsx` with `ConnectionProvider` and `WalletProvider`. Network choice is persisted and triggers reconnects via `NetworkSelector`.
- **Business logic / blockchain helpers:** `solana.js` contains core functions: `issueDocument`, `getHolderDocuments`, `queryBlockchainDocuments`, `claimDocumentAsNFT`, `verifyDocument`, and sharing/import helpers.
- **Rate limiting:** `rateLimiter.js` provides a simple in-process throttler (`globalRateLimiter`) used to avoid Solana RPC 429s during scanning.

Primary data stores & keys (local cache):

- `chaindocs_documents` — array of discovered/issued documents (used as primary UI cache).
- `chaindocs_issuers` — registry of known issuer addresses.
- `chaindocs_claimed` — list of claimed document IDs.
- `chaindocs_nft_<documentId>` and `chaindocs_claim_tx_<documentId>` — per-document claim metadata.

Issuer workflow (presenter view):

- UI: open `IssuerDashboard`. Fill holder public key + metadata and submit.
- Action: `issueDocument(connection, issuerWallet, holderPubKey, documentData)` builds a small JSON payload and writes it as a Memo program instruction in a Solana transaction.
- Post-tx: transaction signature is saved; a `documentRecord` is appended to `chaindocs_documents` and issuer is added to `chaindocs_issuers`. UI reloads documents and shows details/share options.
- Sharing: issuer can generate a base64 `shareableCode` or a `shareableUrl` (via `generateShareableDocumentData`) to send to a holder.

Holder workflow (presenter view):

- UI: open `HolderDashboard`. Connect wallet to enable discovery and claiming.
- Auto-discovery: on connect the client loads cached documents (`getHolderDocumentsFromCache`) immediately, then runs `getHolderDocuments` → which uses `queryBlockchainDocuments` to scan local cache, known issuers' transactions, and recent memos (all rate-limited) to discover documents issued to the holder. Results update `chaindocs_documents`.

- Import: holders can paste a `shareableCode` → `importDocumentFromShareableData` saves the document to localStorage and updates issuer registry.
- Claiming: clicking "Claim as NFT" runs `claimDocumentAsNFT` which writes a Memo `action: 'claim'` tx (simulating mint), stores mock `nftMint` and claim tx in localStorage, and marks the document claimed. UI shows claimed vs unclaimed lists.
- Notifications: `DocumentNotification` displays browser notifications and in-app prompts for newly discovered docs (can trigger claim flow).

Verification flow (presenter view):

- UI: `VerifyDocument` page where user provides Document ID + Credential Hash.
- Action: `verifyDocument(credentialHash, documentId)` checks the cached `chaindocs_documents` for a matching record and compares the hash. If equal → verified; otherwise → invalid. (Note: verification uses local cache, not a trustless indexer in this sample.)

Network & Wallet behavior:

- Network selection (`Devnet`/`Testnet`/`Mainnet`) stored in localStorage and used to compute RPC `endpoint` via `clusterApiUrl`. Changing network reloads the page to reconnect.
- Wallet UI: `WalletConnection` uses `WalletMultiButton`; detects popular extensions (Phantom, Solflare, Backpack) and offers a link to the `WalletGuide`.

RPC scanning strategy & rate limiting:

- `queryBlockchainDocuments` performs three methods: validate local docs on-chain, scan known issuers' recent signatures, and scan recent Memo transactions. Each RPC call goes through `globalRateLimiter` and `rateLimitedRequest` with retries and retry-after heuristics to avoid 429s. This is intentionally conservative (localStorage cache + limited issuer scanning) to keep UX responsive.

UX edge cases & trade-offs (presentable):

- Simplicity: storage and "minting" are simulated via Memo program and localStorage; this is good for demos but not production-grade provenance.
- Performance: blockchain scanning is rate-limited and partial (may return partial results on 429s). A production system would use an indexer or on-chain program to query efficiently.
- Security: credential hash generation is simple (not cryptographically strong in this sample). For production, use secure hashing (e.g., SHA-256) and a proper on-chain program for document registry and NFT minting.