1. Remove any login/signup features you have as **wallet integration** itself acts as **authentication** already. If you wish to not remove it, please make sure you have a definite reason when asked in your presentation.

- No auth screens/routes present (frontend/src checked via rg; none found).

2. As specified in prefinals, your notes app should be able to simulate **building, signing, and submitting txs** to the chain when doing **Create, Update, and Delete** operations in your notes app.

3. This time, instead of sending blank txs, **attach a metadata** to your tx containing all the necessary details you think should be attached. Of course, it is expected that you attach **your actual note content** in the metadata. Example code for that in **JavaScript** below**.** PLEASE READ THE CODE COMMENTS FOR YOUR REFERENCE!!!

4. Since Cardano limits per string in metadata to **64-bytes**, you are **REQUIRED** to implement a "**chunking**" mechanism. You cannot simply attach a long string to the metadata, or the transaction will **fail**. You must use a helper function that detects if a string is too long and splits it into a list of smaller strings.

We implemented this using MeshSDK and KOIOS instead of blockfrost

Transaction build/send via Mesh SDK:

transactionBuilderjs Lines 77-346

export const buildAndSendNoteOperationTransaction

Metadata content + byte-aware 64-byte chunking:

transactionBuilder.js Lines 40-75

const **chunkByBytes**

Metadata includes note content, action, timestamps, flags (labels 674/1337):

transactionBuilder.js Lines 139-179

const messageMetadata

const structuredMetadata

Create/Update/Delete flows use the builder and ledger

WalletContext.jsx Lines 607-840

5. Use the **sendTransaction()** function defined above to send the transaction on-chain, while **simultaneously** storing your notes to your **local database**. You must display the note **immediately** upon clicking save. Querying a database is milliseconds; querying the blockchain history takes time. The database acts as a fast cache.

- In that case, just architect your database to include a **Status** column with an initial value of "**Pending**" so to signify that this note is **yet to be confirmed.** Display this status in your notes app UI also. Done, check Transaction Ledge page. Transactions will have pending status and will update with ADA fees once polled by KOIOS

NotePage.jsx Lines 125-199

Backend Note entity has status , txHash , walletAddress , network, confirmedAt, etc.

Note.java Lines 55-207

Blockchain transactions table mirrors tx info:

BlockchainTransaction.java Lines 17-230

7. **IMPORTANT**: **Implement an Asynchronous Background Worker.** To handle the state synchronization between your local database and the blockchain, you must implement a background process (a worker). This script should run **continuously** (e.g., every **20 seconds**) to check the status of pending transactions.
Polls every 20s, checks pending txs via Koios proxy tx-info, marks ledger confirmed, updates fee, pushes to backend, updates note status to confirmed:

WalletContext.jsx Lines 482 — 545

checkPendingTransactions()

Fee backfill and backend sync

WalletContextjsx Lines 842 — 903

updateTransactionWithActualFee

8. If your database is deleted, then you can just utilize Blockfrost APIs to retrieve your notes from your chain and store it locally. This is the core idea of attaching metadata. The blockchain acts as the **Permanent Source of Truth**, while your local database is merely a **Temporary Cache** for speed.

Function exist, wasn't implemented. Kinda risky.
Reads Koios address-txs + tx-info metadata label 1337 to reconstruct notes (returns list; does not auto-save): WalletContextjsx Lines 931 — 985 restoreNotesFromChain ()