WalletWise — BAI13123 Assignment 2: Detailed Presentation Script

Slide Outline (10–12 Slides)

- Title & Team
  - WalletWise — Personal Finance Tracker
  - Team: Presenter A, B, C, D
  - Module: BAI13123 Assignment 2 (Ionic + Firebase)
- Problem & App Summary
  - Students need a fast, simple way to log income/expenses and see trends.
  - WalletWise provides Ledger, Calendar, and Analytics to understand cash flow.
  - A2 scope: Firebase setup/rules, transactions CRUD with realtime, cache-first loading, clean, reproducible demo.
- Architecture (Ionic + Firebase + Storage)
  - Ionic React UI; modular services and state stores (Zustand).
  - Firebase Auth + Firestore (modular SDK) with persistent local cache.
  - Client cache via Ionic Storage for instant hydration.
  - Pointers: `src/services/firebase.ts:20` , `src/lib/cache.ts:13`
- Data Model (Transactions Focus)
  - Collection path: `users/{uid}/transactions/{txId}`
  - Fields: `{ type: 'income'|'expense', amount:number, date:timestamp, category, subcategory, note?, createdAt, updatedAt }`
  - Types: `Transaction` , `TransactionInput` , `TransactionPatch`
  - Pointers: `src/types/transaction.ts:1` , `src/services/db.ts:71`
- Firebase Setup & Rules
  - Env via Vite: `import.meta.env.VITE_*` (no secrets in repo)
  - Firestore initialized with persistent local cache + multi-tab manager
  - Rules restrict per-user access (`request.auth.uid == uid`) and validate fields
  - Pointers: `src/services/firebase.ts:9` , `src/services/firebase.ts:20` , `firestore.rules:9` , `firestore.rules:36`
- CRUD (Transactions)
  - Create: `addTransaction` + optimistic UI insert in `TxnModal`
  - Read: realtime `onSnapshot` subscription → store updates
  - Update: `updateTransaction` + local patch/revert on error
  - Delete: confirm, optimistic remove, revert on error
  - Pointers: `src/services/db.ts:93` , `src/services/db.ts:175` , `src/components/TxnModal.tsx:178` , `src/pages/Ledger.tsx:133`
- Caching Strategy
  - Cache-first hydration via Ionic Storage on app start
  - Firestore listener refreshes list; cache rewritten with latest snapshot
  - Pointers: `src/state/useTxnStore.ts:52` , `src/state/useTxnStore.ts:96` , `src/lib/cache.ts:13` , `src/lib/cache.ts:19`
- Demo Plan (Checklist)
  - `.env` present; `ionic serve` running; login
  - Show cache-first ledger (offline), then live refresh online

    - Create → instant; Update; Delete + toast; open code pointers live
- Code Quality & Repo Hygiene
    - Typed models; modular structure; sensible query limits
    - Clear README; rules versioned; no secrets
    - Pointers: `README.md:3` , `src/services/db.ts:183` , `firestore.rules:1`
- Results & Next Steps
    - A2 ready: Setup + Rules, CRUD + Realtime, Caching, Clean demo
    - Next: deeper analytics, attachments, more offline/edge testing

4-Presenter Script (6–8 Minutes, Detailed Phrases & Cues)

- Presenter A — Intro (45s)
    - Screen: Dashboard (`/dashboard`)
    - Say: "Hi, we're WalletWise. It's a student-friendly finance tracker to record income and expenses, then view them on a ledger, calendar, and simple analytics. For Assignment 2 we focused on four areas in the rubric: 1) Firebase project setup with environment-based configuration and security rules; 2) transactions CRUD with realtime updates; 3) cache-first loading using Ionic Storage; and 4) a clean, reproducible demonstration."
    - Flash (5s): `README.md:22` (install/run)
    - Say: "We'll keep this to 6–8 minutes and show code along the way."
- Presenter B — Setup & Architecture (75s)
    - Screen: Browser console + Settings (to show user state)
    - Say: "Configuration comes from `.env` via Vite's `import.meta.env` , so no secrets in code. Firestore is initialized with persistent local cache and a multi-tab manager for offline and concurrency."
    - Flash (5–7s each):
        - `src/services/firebase.ts:9` (env keys)
        - `src/services/firebase.ts:20` (persistentLocalCache)
    - Say: "All user data is scoped under `users/{uid}` . Our model centers on transactions with type, amount, date, category, and subcategory."
    - Flash (5–7s): `src/types/transaction.ts:1` (types)
    - Say: "Security rules restrict reads/writes to the signed-in user and validate fields like amount > 0 and date as a timestamp."
    - Flash (5–7s each):
        - `firestore.rules:9` (transactions path)
        - `firestore.rules:36` (validTransaction)
    - Say: "To control reads and cost, we order by date desc and limit to 500."
    - Flash (5s): `src/services/db.ts:180` (query + limit)
- Presenter C — CRUD Demo (3:00)
    - Screen: Ledger (`/ledger`)
    - Create (40s)
        - Do: Click Add → enter amount > 0, date, category/subcategory → Save
        - Say: "We insert optimistically into local state for instant feedback, then call `addTransaction` . The listener confirms and syncs."
        - Flash (5–7s each):
            - `src/components/TxnModal.tsx:139` (validate)
            - `src/components/TxnModal.tsx:186` (optimistic addLocal)

        - `src/services/db.ts:93` (addTransaction)
    - Realtime Read (40s)
        - Do: In a second tab/window, add another transaction; watch this tab update
        - Say: “Realtime uses Firestore’s `onSnapshot` to update our store.”
        - Flash (5–7s each):
            - `src/services/db.ts:175` (subscribeTransactions)
            - `src/state/useTxnStore.ts:96` (subscribe + setItems)
    - Update (40s)
        - Do: Edit the same transaction amount → Save
        - Say: “We patch locally first, then call `updateTransaction`; on error we revert.”
        - Flash (5–7s each):
            - `src/services/db.ts:110` (update)
            - `src/components/TxnModal.tsx:209` (updateLocal before call)
            - `src/components/TxnModal.tsx:219` (revert on error)
    - Delete (30s)
        - Do: Open details → Delete → confirm (watch toast)
        - Say: “We remove optimistically; Firestore confirms; on error we re-add.”
        - Flash (5–7s each):
            - `src/pages/Ledger.tsx:133` (confirm + removeLocal)
            - `src/services/db.ts:128` (delete)

- Presenter D — Caching & Code Quality (90s)
    - Screen: DevTools → Offline → reload Ledger; then go Online
    - Say: “On load, we hydrate from Ionic Storage so the list renders instantly, then the Firestore listener refreshes and rewrites the cache.”
    - Flash (5–7s each):
        - `src/state/useTxnStore.ts:52` (hydrateFromCache)
        - `src/state/useTxnStore.ts:104` (setCached after snapshot)
        - `src/lib/cache.ts:13`, `src/lib/cache.ts:19` (get/set)
    - Say: “The codebase is modular and typed. CRUD depends on a clear service boundary. We cap reads with a `limit(500)` and surface errors via toasts.”
    - Flash (5–7s each):
        - `src/services/db.ts:19` (typed imports)
        - `src/services/db.ts:183` (limit)
        - `README.md:3` (setup), `firestore.rules:1` (versioned rules)
    - Say: “That’s the Assignment 2 scope delivered end-to-end.”
- Presenter A — Closing (15s)
    - Say: “We’ve covered setup and rules, realtime CRUD, cache-first loading, and a clean, reproducible demo. Thank you — happy to take questions.”

Live Demo Choreography (Driver Steps, Expected Results, Fallback)

- Preflight
    - Confirm `.env` present (no secrets on screen). Show keys list: `README.md:11`
    - Start app: `ionic serve`; sign in; console logs project ID: `src/services/firebase.ts:26`
    - Success: app renders; Ledger accessible

- Cache-First Load (offline)
  - DevTools → Network → Offline → reload `/ledger`
  - Expect: list renders from cache; "Loading transactions…" is brief or absent; no errors
  - Source: `src/state/useTxnStore.ts:52` , `src/lib/cache.ts:13`
- Online Refresh (turn online)
  - Disable Offline; wait ≤ 1–2s
  - Expect: any remote changes appear; no errors
  - Source: `src/state/useTxnStore.ts:96` , `src/services/db.ts:175`
- Create Transaction
  - Action: Add → fill → Save
  - Expect: row appears instantly; toast message and no errors
  - Source: `src/components/TxnModal.tsx:186` , `src/services/db.ts:93`
- Update Transaction
  - Action: Edit amount → Save
  - Expect: value changes immediately; persists after refresh
  - Source: `src/services/db.ts:110` , `src/components/TxnModal.tsx:209`
- Delete Transaction
  - Action: Details → Delete → Confirm
  - Expect: row disappears; success toast; on error, item restored
  - Source: `src/pages/Ledger.tsx:133` , `src/services/db.ts:128`
- Code Peek (fast, 5–10s each)
  - Env + Firestore init: `src/services/firebase.ts:9` , `src/services/firebase.ts:20`
  - Realtime subscribe: `src/services/db.ts:175`
  - Cache hydrate/update: `src/state/useTxnStore.ts:52` , `src/state/useTxnStore.ts:104`
  - Modal submit: `src/components/TxnModal.tsx:178`
- Fallback Plan
  - If Firestore fails: switch to Offline to show optimistic UI and caching; explain queued writes and error handling
  - If rules block: show `firestore.rules:9` and confirm auth state; retry after login
  - If network slow: keep second tab realtime demo brief or skip to code pointers

Rubric Mapping Table

- Project & Firebase Setup
  - Evidence: env-driven config (`src/services/firebase.ts:9`), persistent cache (`src/services/firebase.ts:20`), rules in repo (`firestore.rules:1`)
  - Demo proof: show `.env` mapping (`README.md:11`); open code to prove modular SDK and persistence; open rules file
- CRUD (Transactions)
  - Evidence: `add/update/delete/subscribe` in service; UI flows with optimistic UX
  - Files: `src/services/db.ts:93` , `src/services/db.ts:110` , `src/services/db.ts:128` , `src/services/db.ts:175` ; UI: `src/components/TxnModal.tsx:178` , `src/pages/Ledger.tsx:133`
  - Demo proof: create → instant row; second tab add → realtime; edit → value changes; delete → toast/revert
- Caching (Cache-first → Refresh)
  - Evidence: hydrate on start (`src/state/useTxnStore.ts:52`), rewrite cache after snapshot (`src/state/useTxnStore.ts:104`), storage helpers (`src/lib/cache.ts:13` ,

    `src/lib/cache.ts:19`)
  - Demo proof: offline reload shows cached list; online re-enables live refresh and updates cache
- Code Quality & Demonstration
  - Evidence: typed models (`src/types/transaction.ts:1`), modular services, sensible limit (`src/services/db.ts:183`), README and rules
  - Demo proof: quick file peeks; clean, ordered walkthrough with fallbacks and time boxing

Short App Explanation (Lecturer Intro)

WalletWise helps students track income and expenses and see monthly cash-flow trends. For A2, we delivered Firebase setup with environment-based configuration and security rules, complete CRUD for transactions with realtime `onSnapshot` updates, and cache-first loading using Ionic Storage. On start, the app hydrates from cache for an instant list, then refreshes from Firestore and rewrites the cache. The live demo focuses strictly on these areas to match the Assignment 2 rubric.

Q&A Bank (10 Items, With Code Pointers)

- Why Firestore rather than SQLite?
  - Realtime listeners (`onSnapshot`) and per-user scoping reduce backend complexity; local persistence handles offline. See `src/services/db.ts:175` , `firestore.rules:9` .
- How is realtime implemented?
  - A Firestore query subscribes via `onSnapshot` , mapping docs into typed `Transaction` s and updating the store. See `src/services/db.ts:175` , `src/services/db.ts:189` , `src/state/useTxnStore.ts:96` .
- What happens offline?
  - Firestore persistence enables offline reads; we also hydrate from Ionic Storage for instant render and update the cache after the first snapshot. See `src/services/firebase.ts:20` , `src/state/useTxnStore.ts:52` , `src/lib/cache.ts:13` .
- How do rules prevent cross-user access?
  - All data is under `users/{uid}` and rules require `request.auth.uid == uid` ; writes are validated (amount positive, timestamp date). See `firestore.rules:4` , `firestore.rules:9` , `firestore.rules:36` .
- How do you prevent bad inputs?
  - Client validation in `TxnModal` checks amount > 0, valid date/type; server validation in rules rejects invalid payloads. See `src/components/TxnModal.tsx:139` , `firestore.rules:36` .
- What about race conditions in cache refresh?
  - `hydrateFromCache` runs first to set state; the live subscription overwrites and rewrites the cache, so UI eventually reflects canonical Firestore. See `src/state/useTxnStore.ts:52` , `src/state/useTxnStore.ts:96` , `src/state/useTxnStore.ts:104` .
- How do you limit reads and cost?
  - Query uses `orderBy('date','desc')` with `limit(500)` ; errors are handled gracefully with toasts. See `src/services/db.ts:180` , `src/services/db.ts:183` , `src/pages/Ledger.tsx:285` .
- How do you handle errors on update/delete?
  - We optimistically update/remove and revert on error with a toast; delete re-adds the item on failure. See `src/components/TxnModal.tsx:219` and `src/pages/Ledger.tsx:149` .
- Where is auth enforced in code?

    - Service methods read `auth.currentUser` and throw if missing to prevent unauthenticated writes. See `src/services/db.ts:23` , `src/services/db.ts:25` .
- How are dates handled safely?
    - Firestore timestamps map to JS Dates; conversion validated; sort by `date.getTime()` consistently. See `src/services/db.ts:61` , `src/services/db.ts:71` , `src/state/useTxnStore.ts:23` .

One-Page README Insert (Copy-Paste Ready)

- Summary
    - WalletWise (Ionic + React + Firebase). A2 focuses on Firebase setup + rules, transactions CRUD with realtime, and cache-first loading.
- Setup
    - Copy envs: `cp .env.example .env` (see `README.md:5`)
    - Install: `npm install`
    - Run: `ionic serve` (or `npm run dev`)
    - Publish rules: upload `firestore.rules` in Firebase console
    - Files: `src/services/firebase.ts:9` , `firestore.rules:9`
- Demo Steps
    - Login → Ledger
    - Offline: DevTools → Network → Offline → reload; cached list shows (`src/state/useTxnStore.ts:52`)
    - Online: turn Off offline; list refreshes (`src/state/useTxnStore.ts:96`)
    - Create: Add → Save (instant row) → `src/services/db.ts:93`
    - Update: Edit → Save → `src/services/db.ts:110`
    - Delete: Confirm → `src/services/db.ts:128`
    - Realtime: add in 2nd tab; 1st tab updates (`src/services/db.ts:175`)
- Tiny Architecture Diagram (text)
    - Ionic React UI → State Stores (Zustand) → Services (`src/services/firebase.ts` , `src/services/db.ts`)
    - Firebase Auth + Firestore (modular SDK, persistent local cache)
    - Client cache: Ionic Storage for hydration; cache rewrite on snapshot
- Test Caching
    - Toggle offline/online and reload Ledger to observe cache-first then live refresh
- Notes
    - No secrets committed; use `.env.example`
    - Reads capped with `limit(500)` ; rules validate and scope per user