

WalletWise — BAI13123 Assignment 2 Presentation

Plan

Slide Outline (10–12 Slides)

- Title & Team
 - WalletWise — Personal Finance Tracker
 - Team: 4 members (A, B, C, D)
 - Course: BAI13123 Assignment 2 (Ionic + Firebase)
- Problem & App Summary
 - Students need a simple way to record income/expenses and view trends.
 - WalletWise provides a ledger, calendar, and basic analytics.
 - A2 scope: Firebase setup + rules, transactions CRUD with realtime, cache-first loading, clean demo.
- Architecture (Ionic + Firebase + Storage)
 - Ionic React UI; modular services and state stores.
 - Firebase Auth + Firestore (modular SDK, persistent cache).
 - Client cache with Ionic Storage for instant hydration.
 - Pointers: `src/services/firebase.ts:20`, `src/lib/cache.ts:13`.
- Data Model (Transactions Focus)
 - Path: `users/{uid}/transactions/{txId}`.
 - Fields: `{ type: 'income' | 'expense', amount:number, date:timestamp, category, subcategory, note?, createdAt, updatedAt }`.
 - Query: `orderBy('date','desc')`, `limit(500)`.
 - Pointers: `src/services/db.ts:71`, `src/services/db.ts:180`.
- Firebase Setup & Rules
 - Env via Vite (`import.meta.env.VITE_*`), no secrets in repo.
 - Firestore initialized with persistent local cache and multi-tab support.
 - Rules: restrict to `request.auth.uid == uid` + field validation.
 - Pointers: `src/services/firebase.ts:9`, `src/services/firebase.ts:20`, `firestore.rules:9`.
- CRUD (Transactions)
 - Create: `addTransaction` (optimistic insert in UI).
 - Read: realtime `onSnapshot` subscription updates store.
 - Update: `updateTransaction` with local patch.
 - 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 from Ionic Storage on load.
 - Listener refreshes items and rewrites cache.
 - 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.
 - Show cached ledger (offline), then online live refresh.
 - Create → appears instantly; Update; Delete + toast.

- Code peek: `src/services/firebase.ts`, `src/services/db.ts`,
`src/state/useTxnStore.ts`, `src/components/TxnModal.tsx`.
- Code Quality & Repo Hygiene
 - Typed models; modular services/stores/pages; sensible limits.
 - README setup; security rules versioned; no secrets.
 - Pointers: `README.md:22`, `firebase.rules:1`, `src/services/db.ts:183`.
- Results & Next Steps
 - A2 complete: Setup, CRUD, Caching, clean demo.
 - Next: richer analytics, attachments, more offline tests.

4-Presenter Script (6–8 Minutes)

- Presenter A — Intro (45s)
 - Screen: Dashboard summary (`/dashboard`).
 - Lines: “We’re WalletWise, a simple finance tracker to record income and expenses and see monthly trends. For Assignment 2 we implemented four rubric areas: project and Firebase setup, CRUD for transactions with realtime updates, client-side caching for fast loads and offline, and a clean demo.”
 - Flash code (5s): `README.md:22` (dev steps).
- Presenter B — Setup & Architecture (75s)
 - Screen: Settings or console showing project ID log.
 - Lines: “Configuration is env-driven via Vite; no secrets in code. Firestore is initialized with persistent local cache and multi-tab coordination. Our model stores transactions under `users/{uid}` and queries by date with a cap to reduce reads.”
 - Flash code (5s each): `src/services.firebaseio.ts:9` (env), `src/services/firebase.ts:20` (persistence), `firebase.rules:9` (rules), `src/services/db.ts:180` (query).
- Presenter C — CRUD Demo (3:00)
 - Screen: Ledger (`/ledger`).
 - Create: Click Add → fill Amount/Date/Category → Save. “New row appears immediately (optimistic insert), Firestore confirms.”
 - Flash: `src/components/TxnModal.tsx:178`, `src/services/db.ts:93`.
 - Realtime Read: In a second tab, add another transaction (same user). “This tab updates instantly via `onSnapshot`.”
 - Flash: `src/services/db.ts:175`.
 - Update: Edit the same transaction amount → Save. “Local patch then Firestore persist.”
 - Flash: `src/services/db.ts:110`.
 - Delete: Open details → Delete → confirm. “Optimistic remove; toast on success; revert on error.”
 - Flash: `src/pages/Ledger.tsx:133`, `src/services/db.ts:128`.
- Presenter D — Caching & Code Quality (90s)
 - Screen: Ledger offline → online.
 - Lines: “On load, we hydrate from Ionic Storage, then the Firestore listener refreshes and rewrites the cache. This gives instant UX and works offline.”
 - Flash (5s each): `src/state/useTxnStore.ts:52` (hydrate), `src/state/useTxnStore.ts:96` (subscribe/update), `src/lib/cache.ts:13` (get), `src/lib/cache.ts:19` (set).
 - Lines: “Code is modular and typed, with limits to manage cost and performance. Setup is documented and rules are in VCS.”

- Flash: `src/services/db.ts:183` (limit), `README.md:3` .
- Presenter A — Closing (15s)
 - Lines: "That covers A2: secure setup, robust CRUD with realtime, cache-first loading, and a clean demo. Thank you."

Demo Choreography (Checklist + Fallback)

- Preflight
 - Verify `.env` exists and keys match README template: show `README.md:11` .
 - Run `ionic serve`; open console to see project ID log at `src/services.firebaseio.ts:26` .
- Cache-first Load
 - Optional: DevTools → Network → Offline; reload Ledger.
 - Expectation: transactions list renders from cache; loading clears (source: `src/state/useTxnStore.ts:52`).
- Turn Online → Live Refresh
 - Disable Offline; list updates from Firestore; no errors.
- Create Transaction → Instant Row
 - Add, then Save; expect new row immediately; Firestore confirms (source: `src/components/TxnModal.tsx:186`, `src/services/db.ts:93`).
- Update Same Transaction
 - Edit amount; expect visible change without reload (source: `src/services/db.ts:110`).
- Delete With Confirm
 - Confirm delete; expect item removed and success toast (source: `src/pages/Ledger.tsx:133`, `src/services/db.ts:128`).
- Brief Code Peek (5–10s each)
 - Env + Firestore init: `src/services.firebaseio.ts:9`, `src/services.firebaseio.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 (if Firestore hiccups)
 - Toggle DevTools “Offline” and proceed: optimistic UI still shows create/edit/delete locally; explain persistence resumes when back online. Any write error surfaces as toast and reverts (already handled in code).
- Rubric Callout (spoken wrap)
 - Setup & Rules, CRUD, Caching, Code Quality & Demonstration — each proven live with file pointers.

Rubric Mapping Table

Criterion	Evidence (files, UX)	Proof in Demo
Project & Firebase Setup	Env config, modular SDK, rules in repo. Files: <code>src/services.firebaseio.ts:9</code> , <code>src/services.firebaseio.ts:20</code> , <code>firestore.rules:9</code> , <code>README.md:3</code>	Show <code>.env</code> mapping; console prints project ID; open rules; mention persistent cache.
CRUD (Transactions)	add/update/delete/subscribe. Files: <code>src/services/db.ts:93</code> , <code>src/services/db.ts:110</code> , <code>src/services/db.ts:128</code> ,	Create → instant row; second tab add → realtime;

	<code>src/services/db.ts:175</code> ; UI: <code>src/components/TxnModal.tsx:178</code> , <code>src/pages/Ledger.tsx:133</code>	edit → value changes; delete → toast.
Caching (Cache-first → Refresh)	Hydration and cache rewrite. Files: <code>src/state/useTxnStore.ts:52</code> , <code>src/state/useTxnStore.ts:96</code> , <code>src/lib/cache.ts:13</code> , <code>src/lib/cache.ts:19</code>	Offline reload shows cached list; back online refreshes and rewrites cache.
Code Quality & Demo	Typed, modular code and performance limits; README hygiene. Files: <code>src/services/db.ts:19</code> , <code>src/services/db.ts:183</code> , <code>README.md:22</code>	Show imports/types; highlight <code>limit(500)</code> ; ordered, time-boxed walkthrough with fallbacks.

Short App Explanation

WalletWise is a student-friendly finance tracker to record income and expenses, then view them in a ledger, calendar, and simple analytics. For Assignment 2 we delivered Firebase setup via env-based configuration, secure Firestore rules, full CRUD for transactions with realtime subscriptions, and cache-first loading using Ionic Storage. On app start we render from cache and then refresh from Firestore; all changes sync live via `onSnapshot`. The demo focuses strictly on these A2 rubric items.

Q&A Bank (8–10)

- Why Firestore instead of SQLite?
 - Realtime listeners and per-user scoping with minimal backend, plus offline cache. See `src/services/db.ts:175` and `firebase.rules:9`.
- How is realtime achieved?
 - `onSnapshot` subscription updates store on each change. See `src/services/db.ts:175`, wired in `src/state/useTxnStore.ts:96`.
- What happens offline?
 - Firestore persistence + Ionic Storage hydrate the list; writes queue and sync when back online. See `src/services/firebase.ts:20`, `src/state/useTxnStore.ts:52`.
- How do rules prevent cross-user access?
 - All paths are `users/{uid}/...`; rules check `request.auth.uid == uid` and validate fields. See `firebase.rules:4`, `firebase.rules:9`.
- Any race conditions in cache refresh?
 - Hydrate runs first; subscription overwrites state and rewrites cache on next snapshot. See `src/state/useTxnStore.ts:52`, `src/state/useTxnStore.ts:104`.
- How do you limit reads/cost?
 - Query uses `orderBy('date', 'desc') + limit(500)`. See `src/services/db.ts:180`, `src/services/db.ts:183`.
- Client-side validation?
 - Amount > 0 and valid date/type checked in UI. See `src/components/TxnModal.tsx:139`.
- Error handling strategy?
 - Optimistic updates with revert + toast on failure. See delete flow `src/pages/Ledger.tsx:133` and submit flow `src/components/TxnModal.tsx:201`.
- Where do env keys live?
 - `.env` following `README.md:11`; read in `src/services.firebaseio.ts:9`. No secrets committed.

- Auth enforcement in code?
 - CRUD reads `auth.currentUser` and throws if missing. See `src/services/db.ts:23`.

One-page README Insert

- Summary
 - WalletWise (Ionic + React + Firebase). A2 focuses on: Firebase setup + rules, transactions CRUD with realtime, 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 `firebase.rules` in Firebase console.
 - Files: `src/services/firebase.ts:9`, `firebase.rules:9`.
- Demo Steps
 - Login → Ledger.
 - Optional offline: DevTools → Network → Offline → reload; cached list shows (`src/state/useTxnStore.ts:52`).
 - Online: disable Offline; list refreshes from Firestore (`src/state/useTxnStore.ts:96`).
 - Create → Save (instant row) → `src/services/db.ts:93`.
 - Update → Save → `src/services/db.ts:110`.
 - Delete → confirm → `src/services/db.ts:128`.
 - Realtime: add from second tab; first tab updates (`src/services/db.ts:175`).
- Architecture Diagram (text)
 - Ionic React UI → State Stores → Services (`src/services/firebase.ts`, `src/services/db.ts`).
 - Firebase Auth (session) + Firestore (modular SDK, persistent cache).
 - Client cache: Ionic Storage for hydration; Firestore persistence for offline.
- Test Caching
 - Toggle offline/online and reload to observe cache-first render and live refresh.
- Notes
 - No secrets in repo; use `.env.example`.
 - Reads capped with `limit(500)`; rules validate and scope per user.