**Phase 0 — Stabilize Auth & Sessions (foundational)**

**Why now:** Everything else sits on this.

1. **Session & Logout**

* Implement: supabase.auth.getSession() on app load; store user in state; add logout() button.
* Guard: purchase/offer actions open login if no session.
* Done when: refresh keeps user logged in; logout clears UI and blocks purchase/offer.

1. **Guest flow enforcement**

* Gate “Buy” & “Offer price” → if guest, open login modal/page (keep return‑to).
* Done when: guests can browse but cannot complete protected actions.

(Skip magic link & Google for now—see Phase 5.)

**Phase 1 — Data Model & Rules**

**Why now:** Unblocks purchase/offer/notifications.

1. **Tickets: auto‑expire**

* Add status values: active | sold | expired.
* Job: server function or DB cron to set expired where event\_date < now() and not sold.
* Done when: past events no longer purchasable and hidden from “active” list.

1. **Offers (buyer‑specific discounts)**

* Table offers(id, ticket\_id, buyer\_id, amount, status, created\_at).
* Rule: seller accepts/rejects; if accepted, only that buyer can purchase at amount (others see regular price).
* Done when: accepted offer gives buyer a special checkout price; others unaffected.

1. **Transactions (audit trail)**

* Table transactions(id, ticket\_id, seller\_id, buyer\_id, price\_paid, created\_at).
* Done when: each purchase inserts exactly one transaction row.

1. **Balances (“coins”)**

* Keep in auth.users profile or a small wallets(user\_id, balance)—you preferred users table. Seed balance=1000.
* Done when: new users start with coins; balance shows in header/profile.

1. **RLS tightening**

* Tickets: public can read active only; owner can read all; create/update/delete only by owner.
* Offers: buyer can create/read theirs; seller of the ticket can read offers on their tickets; seller/buyer can update status/cancel.
* Transactions: buyer or seller can read their rows.
* Storage: only seller (before sale) and buyer (after sale) can read ticket files; others blocked.
* Done when: public cannot fetch protected files; queries from an unrelated user return 0 rows.

**Phase 2 — Atomic Purchase Flow (core demo)**

**Why now:** This is the heart of Swapit.

1. **Purchase function (atomic)**

* Single RPC or Edge Function:
  + Validates: ticket active, not expired; buyer has coins; optional accepted‑offer match for special price.
  + Deducts coins → marks ticket sold → creates transactions row → (mock) replaces barcode string.
  + All in one transaction; rollback on any failure.
* Done when: two simultaneous purchases can’t both succeed; balances and statuses consistent.

1. **Barcode “mock regeneration”**

* Replace barcode with a new unique string during the same transaction.
* Keep old string visible to seller (read-only history or “last barcode” on ticket) if you want; buyer sees the new one.
* Done when: after purchase, seller cannot access buyer’s new barcode or ticket file.

**Phase 3 — UI for Buying, Offers, and Lists**

**Why now:** Surfaces the core features you’ll demo.

1. **Ticket Details modal**

* Show: image, date, city, price, barcode placeholder, seller name.
* Actions:
  + If logged in & not owner: **Buy** (uses purchase function)
  + **Suggest price**: slider 0 → price; **Submit offer**
* Done when: happy paths for buy & offer work and show confirmations.

1. **Offers inbox (seller)**

* Page: list incoming offers (ticket, buyer, amount, time).
* Actions: Accept / Reject → updates offer + triggers notification.
* Done when: accepting creates a buyer‑specific discounted path to purchase.

1. **My Tickets**

* Tabs: **Selling** (active/expired) and **Purchased** (sold to me).
* Seller can **Update price** for active tickets.
* Done when: lists filter correctly; price updates take effect.

1. **Filters & Sort**

* Replace dropdown with **checkbox groups** (city/category), simple date range, and sorts (price ↑/↓, newest, soonest event).
* Done when: combinations work and persist in the URL/search params.

**Phase 4 — Notifications (in‑app, realtime)**

**Why now:** Makes the demo feel alive.

1. **Notification model**

* Table notifications(id, user\_id, type, payload(jsonb), created\_at, is\_read default false).
* Types: OFFER\_RECEIVED, OFFER\_ACCEPTED, TICKET\_SOLD.
* Write notifications inside the same mutations that cause them.

1. **Realtime delivery**

* Use **Supabase Realtime** subscription on notifications filtered by user\_id to push new items.
* UI: bell icon with badge; dropdown/feed page; click → ticket/offer.
* Done when: seller sees “OFFER\_RECEIVED” instantly; buyer sees “OFFER\_ACCEPTED” instantly.

*(This is what I meant by #18 earlier: Supabase Realtime = websockets/CDC so new rows push to the browser immediately.)*

**Phase 5 — Nice‑to‑have for credibility**

**Do after core demo is solid.**

1. **Google Sign‑In**

* Add OAuth provider in Supabase; button on login page.
* Done when: login via Google returns to app with profile & initial coins.

1. **About page**

* Short mission, how it works, your team, contact.

1. **UI polish**

* Consistent spacing, hover states, disabled states, error toasts; ensure modal scroll for long details.

**Phase 6 — Mobile Readiness & Deploy**

**Why now:** Last mile to show it to students/lecturer.

1. **Responsive pass**

* Test: home, details modal, add‑ticket, login, offers inbox, my tickets at 360–414px width.
* Fix: wrap filters into accordion/drawer on small screens; bigger tap targets.

1. **URL & hosting**

* Choose host (Vercel is easiest). Create .env (move keys out of app.config.ts):
  + NEXT\_PUBLIC\_SUPABASE\_URL=...
  + NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=...
  + (If using Edge Functions) SUPABASE\_SERVICE\_ROLE server‑side only.
* Configure auth redirect URLs; set CORS for Storage/Functions.
* Done when: public URL loads, login works, Storage downloads work.

**Dependencies at a glance**

* Phase 0 → unlocks everything.
* Phase 1 (schema/RLS) → required for Phases 2–4.
* Phase 2 (purchase RPC) → required before notifications for TICKET\_SOLD.
* Phase 3 UI depends on Phases 0–2.
* Phase 4 needs Phase 1 tables + mutations emitting notifications.
* Phases 5–6 are optional/polish but necessary for public demo URL.

**Definition of Done (lecturer‑ready demo)**

* Guest browsing works; buy/offer gated by login.
* A user can list a ticket, another can offer, seller accepts, buyer purchases at discounted price, coins deducted, ticket marked sold, barcode string changes.
* Notifications appear in real time for seller (“offer”) and buyer (“accepted”).
* Expired tickets auto‑hide; purchased tickets appear under “Purchased”.
* Public URL accessible; UI usable on phone width.

**Your immediate next actions (concrete)**

1. **Phase 0**: Add session persistence + logout + guards (1–2 files).
2. **Phase 1**: Create offers, transactions, notifications tables; add ticket expired job; tighten RLS.
3. **Phase 2**: Implement purchase() RPC/Edge Function (atomic deduct → sold → transaction → barcode mock).
4. **Phase 3**: Wire the Details modal to offer & purchase; build Offers inbox; add My Tickets tabs.
5. **Phase 4**: Add realtime notifications (subscribe + bell).