Minimal Viable Product Definition for Flux POS & Inventory System (Trimmed MVP)

Flux is a cross-platform point-of-sale (POS) and inventory system built on **Next.js**, **TypeScript** and **Supabase**. The Charter's core idea is to prove *"true offline"* behaviour, branch-level isolation and essential retail flows first and then expand. This trimmed MVP is the **smallest version that a real shop can run**, while deferring accounting-heavy and nice-to-have modules to later phases.

---

1) MVP feature filter (trimmed)

The table below classifies features from the detailed scope into those **required for the MVP** and those deferred to a **post-MVP (Phase 2)**. It keeps the same structure as the original document but pushes "more-than-enough" features that increase complexity into a later phase.

In MVP (required to prove Flux end-to-end)	Post-MVP (Phase 2+)
PWA + offline framework: manifest and service-worker cache for core UI; local storage (IndexedDB) for an offline queue; auto-sync when online	Background notifications, offline analytics, advanced service-worker caching strategies
User management & security: Supabase Auth login; roles (super-admin, manager, cashier); branch isolation via RLS; user profile + password reset	MFA, bulk user import/export, advanced preference settings, deep audit logging UI
Core masters (minimum set): branches/locations; categories; units; items (basic variants if needed); suppliers; customers (basic credit fields)	Manufacturer master, extra accounting masters (banks, ledger accounts, cash-in/out reasons, etc.)
Inventory (minimum control): batch/expiry tracking; stock on hand per location; **simple stock adjustment** (increase/decrease with a reason note)	**Inventory reconciliation / cycle counts**, complex adjustment reasons framework, serialised assets tracking
Sales/POS (must work offline): touch POS; barcode scan (keyboard-wedge style); product search; **simple discounts**; payments at sale time (cash/card/credit); print receipt via browser print; offline invoice queue + sync	Temporary invoices, invoice post-adjustments, complex promotion engine (loyalty/gift cards), advanced discount rules
Returns (minimal): **sale void/cancel before finalising** + **simple return/refund against an invoice**	Formal "credit note" accounting workflows for strict accounting documents
Purchasing (minimum): create PO; receive GRN; update stock automatically; PO status tracking	GRN post-adjustments, **posted GRN cancellation**, separate purchase return invoice module, supplier payment processing
Multi-location (minimum): transfer-out note + transfer-in receipt; stock balance per location	Advanced receive notes separate from transfers, transit optimisation, complex transfer adjustments
Basic reporting (minimum): sales summary (by date/user/branch); stock balance/valuation; expiry warnings; outstanding credit summary	Full financial statements (trial balance, income statement, balance sheet, cashflow), day/shift/month end packs, full audit trails, detailed ageing packs

Key trims (what was pushed out compared to the original MVP)

These items often slow down solo builds and therefore have been moved to Phase 2+:

- **Barcode/label printing** → moved to Post-MVP (receipt printing only in MVP)
- **Inventory reconciliation / cycle count** → moved to Post-MVP (keep simple stock adjustments only)
- **Posted GRN cancellation + post-adjustments** → moved to Post-MVP (MVP supports draft/edit before posting + basic returns)
- **Fast/slow moving + advanced ageing packs** → moved to Post-MVP (keep only the minimum summaries)

All of the above remain documented but are not part of the first release.

---

2) MVP product backlog (Epics & user stories) — trimmed

This section keeps the same "Epic list" style but removes user stories for the pushed-out features. Each epic remains aligned with the roles described in the charter (super-admin, store manager, cashier).

Epic 1 – PWA / offline infrastructure

- **Installable PWA from browser** — users can install the app from their browser.
- **App loads offline using cached assets** — the application should load quickly and display cached data when offline so that work can continue during network outages.
- **Queue unsynced transactions in IndexedDB** (with timestamps and UUIDs) and retry later.
- **Auto-sync when connectivity returns** — unsynced transactions should synchronise automatically when back online so that Supabase data stays consistent.
- **Show "pending sync count" to the user** — display a badge indicating the number of unsynced transactions.

Epic 2 – Authentication & branch security

- **Login with Supabase Auth** — email and password login ensures a secure session.
- **Roles**: super-admin / manager / cashier — role-based permissions restrict access to modules and branches.
- **Assign users to a branch** — enforce branch isolation via RLS policies.
- **Profile update + password reset** — users can maintain their account details.

Epic 3 – Core inventory masters

- **Create/edit categories, units, items and locations** — reflect actual products and warehouses.
- **Create/edit suppliers and customers** (basic credit fields) — manage purchasing and credit sales.
- **Batch/expiry fields for items** (where needed).
- **Maintain stock per location** — track stock on hand by branch.

Epic 4 – Sales & POS (offline mandatory)

- **POS screen** — scan/search, add items, edit quantity, simple discount.
- **Payments** — accept cash/card/credit at checkout.
- **Print receipt** via the browser's print API.
- **Offline sale** — save the invoice locally, sync later and update stock after sync.

Return/void stories (minimal):

- **Void a sale before finalising** — allow cancellation of a sale prior to completion.
- **Return/refund against an invoice** — support a simple workflow for returns and refunds.

Epic 5 – Purchasing (minimum)

- **Create Purchase Orders (PO)** — define expected quantities linked to suppliers.
- **Record Goods Received Notes (GRN)** — create GRN from a PO or directly.
- **Stock updates on GRN** — update stock automatically when items are received.
- **Track PO status** (open/partial/closed).

In this MVP: a GRN can be edited or deleted while in "draft". Once posted, no cancellation — that functionality moves to Phase 2.

Epic 6 – Multi-branch operations

- **Create transfer-out note from Branch A**.
- **Record transfer-in receipt at Branch B**.
- **Stock balances update per location**.
- **View balances/valuation by location** — see stock balances and valuations per branch.

Epic 7 – Basic reporting

- **Sales summary by date/user/branch**.
- **Stock balance + stock valuation**.
- **Expiry warning list**.
- **Outstanding credit summary** (simple totals and list).

---

3) Walking skeleton (Sprint 1 goals) — still valid, slightly tighter

Your current "walking skeleton" section remains applicable. This trimmed version focuses on proving the whole pipe with the least feature set:

- **Next.js + TypeScript project** with PWA manifest and basic service-worker cache.
- **Supabase project connected** with row-level security enabled.
- **Login + branch restriction** (one branch for now).
- **Create one category, unit and item** in Supabase.
- **Cache items locally**.
- **Minimal POS** — sell that item (cash).
- **Offline sale** — save sale to IndexedDB (*synced = false*).
- **When online** — sync sale → update stock → show sale in a simple report list.
- **Receipt print** — optional in Sprint 1 but include early if printer risk is high.

This skeleton exercises all major layers: PWA installability, offline storage and sync, Supabase Auth/RLS, CRUD operations and a simple report. It validates that the chosen tech stack meets the trimmed MVP definition.

---

4) Technical “gotchas” for the MVP

The same risks identified in the original document still apply; they are summarised here to align with the trimmed scope:

- **Offline sync correctness** — each offline record stored in IndexedDB must have a stable primary key or UUID to avoid duplicates when syncing. Timestamps enable a simple “last-write-wins” policy. Ensure that any triggers on Supabase (e.g., adjusting stock on sales/GRN events) execute correctly when batched records arrive.
- **Enforcing RLS offline** — row-level security (RLS) is enforced on the server side, but when offline the client must only cache and manipulate data for the user’s branch. The local store should be namespaced by *location_id*, and the sync service must include this identifier in every request to satisfy Supabase policies. Failing to do so could leak data between branches when the device goes back online.
- **Service-worker integration with Next.js** — adding a custom service worker to a Next.js project requires careful configuration. Incorrect caching strategies may serve stale data or cause unexpected behaviour when new deployments are released. Dynamic API requests should not be cached indiscriminately, and updates to the service worker need proper versioning and an update flow to avoid “white screen” issues on clients.

Addressing these risks early in the MVP ensures that the offline-first architecture is robust and secure before expanding to more complex financial features in later phases.

Summary recommendation

✅ This trimmed MVP is **safer, faster and more realistic** for a solo build. It still proves Flux’s key promise — **true offline POS**, **branch isolation**, **stock correctness** and **basic operations** — without pulling you into accounting-grade complexity too early.