ChatGPT

Project Charter – Flux POS & Inventory Management System

Executive Summary

Flux is a cross-platform point-of-sale (POS) and inventory management application built as a Progressive Web App (PWA) using **Next.js**, **TypeScript**, **Tailwind CSS**, and **Supabase**. It helps small and medium-sized retail businesses consolidate sales, purchasing, stock control and reporting under one system. The system runs in any modern browser or as an installable app on desktop and mobile. PWA technology allows updates to be delivered instantly and supports native-like features such as home-screen installation and push notifications [1]. Core functionality, such as ringing up sales, scanning barcodes and updating stock, must continue to work while offline, with data synchronised to the cloud once connectivity is restored.

The business value of Flux lies in eliminating fragmented paper-based processes and disparate software. With a central database, managers gain real-time visibility into stock levels across multiple branches, reducing stock-outs and excess inventory [2]. Automation of barcode scanning and order processing improves accuracy [3], while flexible reporting aids financial control and regulatory compliance. Because Flux targets small retailers, the system emphasises ease of use and quick deployment without the cost and complexity of native mobile apps.

Vision & Goals

Vision

Create a reliable, offline-first POS and inventory platform that scales from a single store to multiple locations. The application will deliver a fast, responsive user experience comparable to native apps, work seamlessly in low-connectivity environments, and provide comprehensive management reporting.

Definition of Done for the MVP

1. **PWA functionality** – Flux must be installable on mobile and desktop with a web-app manifest. A service worker caches assets and implements a cache-first strategy for critical resources [4], enabling the app to load quickly and operate offline. Lighthouse performance and accessibility scores should be in the green range (>90) [5].
2. **User management & security** – User roles and permissions are defined in Supabase with row-level security (RLS). RLS must be enabled on all exposed tables [6] to ensure each user only accesses data for their branch. Authentication is handled through Supabase Auth.
3. **Core inventory management** – Users can create and edit categories, suppliers, items, units, and stock locations. Stock quantities are tracked per location with batch, expiry, and unit conversions.
4. **Sales/POS** – Cashiers can create sales invoices via a touch-friendly interface, scan barcodes, apply discounts, and accept payments (cash, card, credit). Sales must work offline; transactions are queued locally using IndexedDB [7] and synced when online [8].
5. **Purchasing** – Users can create purchase orders and goods-received notes (GRNs), record returns, and adjust stock. Supplier details are stored and purchase documents update inventory automatically.

6. **Multi-location support** – Stock is maintained per branch. Users can initiate stock transfers between locations and see location-specific balances. Branch-level reporting is available.
7. **Reporting** – Basic dashboards provide sales summaries, stock valuation, ageing reports for customers and suppliers, and financial statements.

Additional Goals

- Deliver a single codebase that runs on web, Android and iOS via PWA, avoiding separate native apps.
- Ensure offline transactions and inventory updates sync automatically when the network returns, resolving conflicts using a "last-write-wins" policy with future improvements for more advanced reconciliation [9].
- Achieve high performance and accessibility: Lighthouse scores >90 (green) across performance, accessibility and best practices [5].
- Ensure scalability by using Supabase's serverless Postgres backend with Row-Level Security and multi-tenant design, supporting unlimited branches and users.

Scope of Work (Functional Epics)

The PDF outlining the menu structures and screen layouts was analysed to derive the following high-level epics:

1. User Management & Security

- **User Roles & Permissions** – Create roles (super-admin, manager, cashier) and assign permissions for each module. Configure Supabase RLS policies so every request is filtered by branch or role [6].
- **Authentication & Session Management** – Implement login, password reset, and multi-factor authentication. Tokens expire and refresh automatically.
- **User Profile & Preferences** – Users can update their details, password, and language/theme preferences.

2. Inventory Management

- **Item Master Data** – CRUD interfaces for categories, manufacturers, items (with variants), units, and batch/expiry details.
- **Suppliers & Customers** – Manage supplier and customer records, including contact details, credit limits and days.
- **Stock Adjustments** – Record stock adjustments and reconcile inventory counts. Support barcode printing for items.

3. Sales & POS

- **Sales Invoice** – Touch-friendly POS screen with barcode scanning, product search, multi-payment methods, and customer selection. Supports temporary invoices, credit notes, credit invoices and invoice cancellation.
- **Offline Sales** – Sales are stored in IndexedDB while offline [7] and synced later [8]. The UI shows pending transactions.
- **Receipt Printing** – Generate receipts/invoices with custom templates; support small thermal printers via the browser's print API.

4. Purchasing

- **Purchase Orders (PO)** – Create, edit and approve POs. Link to suppliers and update expected quantities.

- **Goods Received Note (GRN)** – Record receipts of goods, handle returns and cancellations, and adjust stock accordingly. Support GRN post adjustments.

5. Multi-Branch Operations

- **Location Management** – Define branches/locations and assign stock. Users belong to specific locations but can have cross-location roles.
- **Stock Transfers** – Create stock transfer notes and receive notes to move goods between locations. Ensure inventory balances update correctly.
- **Location-wise Reporting** – Provide stock balance and valuation by location [2]. Generate location-specific sales and purchase summaries.

6. Financials & Accounts

- **Cash Management** – Record cash in/out transactions with reasons. Track shifts and day-end cash counts.
- **Ledger & Journals** – Create journal entries, general ledger accounts and bank accounts. Support closing periods (day, shift, month, year).
- **Payments** – Process customer receipts, supplier payments and adjustments. Manage cheques (deposit, return, fixed deposit).

7. Reporting

- **Stock Reports** – Item lists, fast-moving/slow-moving items, stock valuation, expiry warnings and stock movement history.
- **Sales Reports** – Invoice listings, sales summaries by date and user, customer outstanding and ageing reports.
- **Purchase Reports** – GRN listings, summaries, outstanding GRNs.
- **Financial Reports** – Trial balance, income statement, balance sheet, cash-flow reports, and day/ shift/month end summaries.

Technical Architecture & Non-Functional Requirements

High-Level Architecture

- **Frontend (PWA)** – Built with Next.js/React and TypeScript. The app uses Next.js App Router and the built-in PWA guide: create a `manifest.ts` or `manifest.json` file specifying the app name, start URL, display mode and icons [10]. Service workers handle asset caching and offline behavior, using a cache-first strategy for core assets and IndexedDB for data storage [4] [7].
- **Local Data Storage** – Use **IndexedDB** via the `idb` library to store sales, inventory changes and other transactions locally. IndexedDB supports structured objects, large storage (hundreds of megabytes) and asynchronous operations [11], making it superior to `localStorage` (which is limited to ~5-10 MB and synchronous). Data objects include a `synced` flag and timestamps.
- **Sync & Conflict Resolution** – Implement a sync service that scans for unsynced records and pushes them to Supabase when `navigator.onLine` is true [8]. Use the Background Sync API where supported to retry queued tasks automatically [12]. For conflicts, adopt a last-write-wins policy initially; later phases can implement more sophisticated merging [9].
- **Backend (Supabase)** – Supabase provides a managed Postgres database, real-time subscriptions and RESTful API. Supabase Auth handles user authentication. Row-Level Security (RLS) must be enabled on all public tables [6], with

policies ensuring users only access records for their branch. Triggers ensure data integrity (e.g., adjusting stock on sales/GRN events).

- **Multi-Branch Schema** – Include a `location_id` on stock and transaction tables. Use RLS to restrict queries by location and role. A centralised design allows reporting across locations [2] while maintaining isolation.

Non-Functional Requirements

1. **Performance** – PWA must achieve Lighthouse scores in the "Good" range (>90) across performance, accessibility and best practices [5]. Use code-splitting and lazy loading; compress assets and images; minimize JavaScript. Caching strategies should minimise network requests.
2. **Scalability** – The architecture must support additional branches and concurrent users without degradation. Supabase's serverless Postgres and real-time engine can horizontally scale. Data tables should have indexes on `location_id` and other foreign keys.
3. **Security** – Use HTTPS for all communications [13]. Apply RLS and role-based permissions. Store secrets (API keys) securely. Validate input and sanitize outputs to mitigate injection attacks.
4. **Reliability** – Offline mode must queue transactions reliably. Sync code must handle network failures gracefully and retry. Data integrity across branches must be maintained.
5. **Accessibility** – Follow WCAG 2.1 guidelines. Ensure keyboard navigation, screen-reader labels and sufficient colour contrast. Provide offline status indicators.
6. **Maintainability** – Use TypeScript for type safety and consistent code style. Document architecture and code. Write automated unit and integration tests.

Stakeholders & Roles

Stakeholder	Role/Responsibilities
Product Owner (You)	Defines requirements, prioritises backlog, accepts deliverables, ensures the product meets business needs.
Solo Developer (You)	Acts as Scrum Master and development team. Implements features, writes tests, deploys the application, manages DevOps.
Super Admin	User role responsible for system configuration (locations, roles, preferences), manages other users, and accesses all reports.
Store Manager	Oversees inventory and purchasing at a specific branch; manages staff, reviews daily sales, performs stock transfers and audits.
Stakeholder	Role/Responsibilities
Cashier/User	Performs sales transactions, processes returns, receives stock. Limited to branch-level functions.
Accountant	Reviews financial reports, approves journal entries, closes periods, and manages ledgers and bank reconciliations.

Risks & Mitigation Strategies

Risk	Impact	Mitigation
Solo development bottlenecks & burnout	With one developer, illness or overload can halt progress and quality suffers.	Use Agile time-boxing and prioritise highest-value features for the MVP. Automate testing and continuous integration to reduce manual workload. Consider recruiting freelance help or scheduling buffer time for learning and recovery.
Offline sync conflicts	If transactions occur simultaneously on multiple devices while offline, data conflicts can arise upon sync.	Implement a transaction queue with timestamps; adopt last-write-wins policy initially and log conflicts for manual review [9]. Future versions can implement versioning or merge strategies.
Browser storage limitations	Browsers limit local storage. Using `localStorage` (5– 10 MB) would be inadequate for sales data [14].	Use IndexedDB for structured storage with much higher limits [11]. Periodically purge old data after successful sync.
Security vulnerabilities	Misconfigured RLS or missing HTTPS could expose data.	Enable RLS on all tables [6], write policies for each role, enforce HTTPS [13], and follow OWASP best practices.
Performance degradation	PWA performance may suffer due to heavy bundles.	Use code-splitting, optimize images, and monitor Lighthouse scores, aiming for 90+ performance [5].
Scope creep	Additional features may delay delivery.	Clearly define MVP scope, maintain a prioritised backlog, and re-estimate tasks regularly.

Success Criteria

1. **Offline Reliability** – Sales and inventory transactions created while offline are processed correctly and synchronised when connectivity is restored with no data loss or duplication. At least 95 % of test cases should pass in offline/online toggling scenarios.
2. **Performance Scores** – The PWA scores 90 in Lighthouse performance, accessibility and best practices audits [5].
3. **Data Accuracy Across Locations** – Stock balances and financial reports remain consistent across branches, with discrepancies <1 % during testing.
4. **User Adoption & Usability** – Cashiers and managers can complete common tasks (sales, GRNs, stock transfers) within a target time (e.g., <2 minutes per transaction) and report high satisfaction in user surveys.

5. **Scalability** – System supports at least 5 branches and 20 concurrent users with response times under 500 ms for key operations during load testing.

Conclusion

This project charter serves as a blueprint for developing **Flux**, an offline-capable POS and inventory management PWA tailored for multi-branch retail businesses. By leveraging Next.js, Supabase and service-worker-driven offline strategies, the project aims to deliver a robust, scalable and user-friendly solution that brings real-time insight and operational efficiency to the retail sector. Through phased development, careful risk mitigation and clear success metrics, the solo developer can deliver a valuable product while managing complexity and ensuring quality.

Caution

The project phases and MVP scope are currently being finalized. Consequently, specific release schedules and sprint cycles are excluded from this Charter, which serves primarily to outline the fundamental concept of the application.

---

1 10 Guides: PWAs | Next.js
https://nextjs.org/docs/app/guides/progressive-web-apps

2 3 Multi-Location Inventory Management Guide | Netstock
https://www.netstock.com/blog/streamlining-multi-location-inventory-management/

4 12 Offline and background operation - Progressive web apps | MDN
https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/Offline_and_background_operation

5 Lighthouse performance scoring | Chrome for Developers
https://developer.chrome.com/docs/lighthouse/performance/performance-scoring

6 Row Level Security | Supabase Docs
https://supabase.com/docs/guides/database/postgres/row-level-security

7 8 9 11 13 14 Build a Next.js 16 PWA with true offline support - LogRocket Blog https://blog.logrocket.com/nextjs-16-pwa-offline-support/