ChatGPT

Flux Development Roadmap

This roadmap translates the **Project Charter**, **MVP Definition** and **Full-Scope** documents into a practical development plan for a solo developer. It breaks the work into incremental phases: a walking skeleton that proves the architecture, the trimmed MVP, post-MVP operational enhancements, a financial/accounting expansion and finally advanced analytics. Each phase lists goals, functional features, implementation steps and the specific screens/pages from the full scope that belong to that release.

Overview

Flux is a cross-platform PWA built with Next.js, TypeScript and Supabase. The core vision is to deliver a retail POS and inventory platform that continues to operate offline and scales from a single shop to multiple branches. The Project Charter emphasises PWA installability, offline data using IndexedDB, and multi-branch row-level security (RLS)【237113286733124†L34-L50】. The MVP Definition trims the scope to the smallest set of features a real shop can operate with (stock masters, simple POS, basic purchasing and reporting) and defers nice-to-have features (barcodes, cycle counts, complex promotions) to later phases. The Full-Scope document lists every screen and field required for the final product, including complex accounting modules (ledger, journals, cheques, financial reports) and a comprehensive suite of operational reports.

Guiding principles

- **Incremental delivery:** Build a walking skeleton first to validate the offline architecture and Supabase integration. Then deliver a trimmed MVP, followed by enhancements and eventually the full scope.
- **Offline-first:** Every transactional feature must work when offline by storing records in IndexedDB and syncing when connectivity returns. Local data must be namespaced by `location_id` to enforce RLS even when offline.
- **Solo-dev friendly:** Each phase focuses on a coherent set of epics so that one developer can design, implement and test features without becoming overwhelmed.

Phase 0 – Walking Skeleton (Infrastructure & Proof of Concept)

Goal

Validate the end-to-end stack (Next.js PWA, Supabase RLS, IndexedDB offline storage, basic sync) by implementing the simplest vertical slice. This sprint proves that a dummy item can be sold while offline and synced later.

Key features (functional)

Feature	Source & rationale
Installable PWA with basic service worker and manifest	The charter requires the app to be installable on mobile/desktop with a manifest and a service worker caching critical assets. The walking skeleton should implement a cache-first strategy so the shell loads offline.
Supabase setup with RLS and one branch	Connect to a Supabase project, enable row-level security and create minimal tables (`branches`, `categories`, `items`, `units`) with a `location_id`. Only a single branch is used initially.
Authentication & branch restriction	Implement login using Supabase Auth for one user and enforce branch isolation through RLS.
CRUD for a dummy category, unit and item	Create one category, one unit and one item in Supabase to exercise CRUD and caching.
Minimal POS – sell one item	Build a simple POS interface that lists the dummy item, allows the cashier to add it to the cart and take a cash payment.
Offline sale and sync	Store the sale in IndexedDB with a `synced` flag; when connectivity returns, sync to Supabase and update the stock on hand.
Simple sales report	After syncing, display the sale in a basic report list to confirm the data flow end-to-end.
Optional early receipt printing	The charter notes that receipt printing can be added early if there is risk around printing.

Technical implementation steps

1. **Project scaffolding:** Initialize a Next.js project with TypeScript and Tailwind CSS. Configure the PWA manifest and register a custom service worker that precaches the app shell (HTML/CSS/JS) and sets up a cache-first strategy for core assets. Use the idb library for IndexedDB operations.
2. **Supabase provisioning:** Create a Supabase project. Define `branches`, `categories`, `units` and `items` tables. Include `location_id` columns and enable row-level security policies restricting data to the current branch and role. Configure Supabase Auth for email/password login.
3. **Authentication flow:** Build login and logout pages using Supabase Auth. Store the authenticated user and branch in context. Enforce branch restriction client-side by only caching data for that branch.
4. **Simple master data forms:** Create basic forms to create and edit a category, unit and item. Use Supabase client APIs for create/update and store items in IndexedDB. Display a simple list of items from the local cache.
5. **POS proof-of-concept:** Develop a minimal POS page with a product list, quantity selector and cash checkout. On payment, insert a sale record into IndexedDB (including timestamp, amount and `synced=false`).

6. **Sync service:** Implement a background sync service that runs when `navigator.onLine` is true. It scans IndexedDB for unsynced records and posts them to Supabase; once the server confirms, mark them as synced. Handle duplicates using UUIDs.
7. **Basic reporting:** After syncing, query Supabase for recent sales and display them in a simple report page. Verify that the stock on hand is updated correctly.
8. **Receipt printing (optional):** Use the browser's print API to render a receipt and trigger printing if required.

Screens/Pages in Phase 0

- **Login page** – email/password form using Supabase Auth (aligns with `User` in full scope).
- **Dummy master data pages:** minimal pages for **Category**, **Unit** and **Item** allowing create/edit (from full scope section 1.1 Category and 1.8 Units).
- **POS prototype:** a very simple **Sales Invoice** screen (full scope 2.1.1) that lists the single item and accepts a payment.
- **Simple sales report** – list of synced sales transactions (a subset of the `Invoice Listing` report from full scope 6.2.1).

By the end of Phase 0 you have verified that the stack can support PWA installability, offline storage/sync and RLS enforcement.

Phase 1 – Trimmed MVP (Release 1)

Goal

Deliver a usable POS and inventory system for a single shop or small chain based strictly on the "In MVP" column of the MVP Definition. This release must support day-to-day sales, stock control, basic purchasing and minimal reporting while operating reliably offline.

Key features (functional)

Feature	Source & notes
PWA & offline framework improvements	Continue to refine the service-worker cache and IndexedDB queue. Auto-sync unsynced transactions when online and show a pending sync count.
User management & security	Implement login with Supabase Auth; create roles (super-admin, manager, cashier); assign users to branches; allow users to update their profile and reset passwords. These roles and RLS policies are mandated by the charter.
Core masters	Provide CRUD interfaces for branches/locations, categories, units, items (with variants and batch/expiry fields), suppliers and customers with basic credit limits/days. Manage stock per location.
Inventory control (minimum)	Track stock on hand per location and allow simple stock adjustments (increase/decrease) with a reason note. Batch and expiry tracking are included.

Feature	Source & notes
Sales/POS	Build a touch-friendly POS screen that supports barcode scanning (keyboard wedge), product search, quantity editing, simple discounts and payments (cash, card, credit). Sales must work offline by queuing invoices and updating stock on sync. Include receipt printing via the browser's print API.
Returns/void (minimal)	Allow voiding a sale before finalising and processing a simple return/refund against an invoice.
Purchasing (minimum)	Create purchase orders (POs), record goods received notes (GRNs) and automatically update stock. Track PO status (open/partial/closed). GRNs can be edited or deleted while in draft; once posted, no cancellation.
Multi-location transfers	Create transfer-out notes and transfer-in receipts to move stock between branches and maintain stock balances per location.
Basic reporting	Provide sales summary by date/user/branch, stock balance/valuation, expiry warning list and outstanding credit summary. These correspond to the full scope's simple sales and stock reports.
Maintain RLS and offline security	Ensure that all cached data is namespaced by branch and that the sync service includes `location_id` in requests to satisfy Supabase policies.

Technical implementation steps

1. **Expand database schema:** Build out the full set of MVP tables: `locations`, `items` (with batch/expiry fields), `units`, `suppliers`, `customers` (with credit limit and days), `purchase_orders`, `grns`, `stock_transfers`, `stock_adjustments`, and `sales_invoices`. Add triggers in Supabase to adjust stock on GRN and sale events as described in the charter.
2. **RLS policies & roles:** Define RLS policies for each table so users only access data for their branch and role. Create role-based UI guards in Next.js (super-admin can manage all branches; manager can access own branch; cashier has POS access only).
3. **Master data UI:** Develop pages for **Category**, **Supplier**, **Customer**, **Item** (with batch/expiry fields), **Location** and **Units** using CRUD forms. Save changes offline first then sync. Reference full-scope fields (e.g., category number, name, shortcode; customer credit limit and days).
4. **Inventory module:** Implement stock-on-hand calculations per location. Provide a **Stock Adjustment** page for simple increase/decrease with reason.
5. **POS module:** Build a robust POS component that supports barcode scanning via input focus, product search, quantity editing, simple discounts and multiple payment methods. Integrate offline queuing, receipt printing and sync logic.
6. **Returns/refunds:** Create flows to void an invoice before posting and to process a return/refund by referencing an existing invoice.
7. **Purchasing:** Create pages to create/edit **Purchase Orders** and **GRNs**. When posting a GRN, update stock quantities and set GRN status. Allow editing while the GRN is in draft but forbid cancellation after posting.
8. **Multi-location transfers:** Implement **Transfer-Out** and **Transfer-In** pages. On transfer creation, decrease stock from the source branch; on receipt, increase stock at the destination.

9. **Reporting:** Build simple reporting pages that query Supabase/IndexedDB for sales summaries, stock balances, expiry warnings and outstanding credits.
10. **Sync & conflict handling:** Enhance the Phase 0 sync service to handle new document types (PO, GRN, stock transfer, stock adjustment) and update the `synced` status. Continue using a last-write-wins policy; plan for conflict resolution improvements in later phases.

Screens/Pages in Phase 1

The trimmed MVP corresponds to the following screens from the Full-Scope document:

- **Master data:** Category (1.1), Supplier (1.2), Item (1.4), Location (1.6), Customer (1.7), Units (1.8).
- **Transactions:** Sales Invoice (2.1.1); Return/Refund (simplified subset of 2.1.2 Credit Note and 2.1.5 Credit Invoice); Purchase Order (2.4.5); GRN (2.2.1); Transfer-Out/Transfer-In (2.4.1, 2.4.2); Stock Adjustment (2.4.3).
- **Reports:** Sales summary (3.6 Sales Summary), Stock balance & valuation (6.1.8–6.1.13), Expiry warnings (6.1.18–6.1.19), Outstanding credit summary (6.4.1–6.4.9).
- **Settings:** Basic user profile and password reset (from Tools section 5.2–5.3).

Phase 2 – Operational Enhancements (Post-MVP)

Goal

Enhance operational capabilities beyond the trimmed MVP by delivering features that were explicitly deferred to “Post-MVP (Phase 2+)” in the MVP Definition. This phase makes the system more powerful for retail operations without yet diving into heavy accounting.

Key features (functional)

Feature	Source & notes
Barcode and label printing	Add the ability to print barcodes/labels for items and documents. This feature was intentionally pushed to Post-MVP and corresponds to the full-scope `Barcode Printing` module.
Inventory reconciliation & cycle counts	Implement full inventory reconciliation/cycle count process with variance reports and approval flow, expanding beyond simple adjustments. Corresponds to full-scope `Inventory Reconciliation`.
Advanced stock adjustments & serialised assets	Support complex adjustment reasons, serialised asset tracking and audit trails. Add master tables for adjustment reasons (1.9 Invoice Adjustment Reason, 1.10 GRN Adjustment Reason).
Manufacturer and extra masters	Introduce **Manufacturer** master (1.5) and additional accounting masters such as **Banks and Branches** and **Ledger Accounts** (1.13–1.14). These were deferred from the MVP.
Promotions & complex sales flows	Add temporary invoices, invoice post-adjustments, formal credit notes, credit invoices and more sophisticated discount/promotion rules.

Feature	Source & notes
Purchase enhancements	Enable GRN post-adjustments and cancellation, separate **Purchase Return** module and supplier payment processing.
Advanced transfers & logistics	Support separate receive notes for transfers, transit optimisation and complex transfer adjustments.
Customer & supplier payment modules	Implement customer receipts, supplier payments, payment adjustments and cancellations. Basic receipts were deferred in the MVP; here they become first-class features.
Enhanced user management	Add multi-factor authentication (MFA), bulk user import/export, advanced preference settings and audit log UI.
Background notifications & offline analytics	Implement background sync notifications (e.g., pending transactions) and offline analytics dashboards. Also upgrade service-worker caching strategies for dynamic data.
Advanced reporting & ageing	Provide fast/slow moving item reports and advanced ageing reports for customers and suppliers.

Technical implementation steps

1. **Label/barcode printing:** Design a barcode label template using a library like `jsbarcode`. Create a printing component that opens a browser print dialog for the label. Provide configuration for label size and printer selection.
2. **Cycle count & reconciliation:** Add tables for `cycle_count_sessions` and `cycle_count_lines`. Create a workflow where users select items to count, enter physical counts, compute variances and post adjustments. Support offline counting and sync results when online.
3. **Advanced stock adjustments:** Expand the `stock_adjustments` schema to include reason codes (linking to `invoice_adjustment_reasons` and `grn_adjustment_reasons`), serial numbers and audit trails. Update the UI to capture these fields and generate audit logs.
4. **Additional masters:** Implement CRUD pages for **Manufacturer**, **Banks and Branches**, **Ledger Accounts** (1.13–1.14) and other extra masters (cash in/out reasons). Ensure RLS policies apply.
5. **Complex sales flows:** Build pages for **Credit Note**, **Invoice Post Adjustment**, **Invoice Cancellation**, **Credit Invoice** and **Temp Invoice**. Implement logic to adjust stock and outstanding balances accordingly. Add a promotion engine with percentage/flat discounts and a loyalty/gift card framework.
6. **Purchase returns & GRN management:** Create modules for **Purchase Return Note**, **GRN Post Adjustment** and **GRN Cancellation**. Implement supplier payment processing: allow recording of supplier payments, adjustments and cancellations.
7. **Transfer enhancements:** Add support for **Stock Receive Note** (receiving goods from transfers), transit status tracking and complex transfer adjustments. Add a transit stock ledger to prevent negative balances while in transit.
8. **Payment modules:** Build pages for **Customer Receipt**, **Supplier Payment**, **Customer/Supplier Payment Adjustment** and **Cancellation**. Ensure payments update customer/supplier outstanding balances.

9. **User management upgrades:** Integrate MFA using Supabase's OTP/email features, build a CSV import/export for users, add preferences and theme settings and display an audit log UI for user actions.
10. **Background services & analytics:** Implement background notifications (e.g., using the Background Sync API) to alert users to unsynced transactions or stock variances. Develop offline dashboards that compute simple analytics from IndexedDB and display them even without network. Optimise caching strategies in the service worker.
11. **Advanced reporting:** Develop new reports: fast moving items, non-moving items, detailed ageing by item/customer/supplier, stock movement by item, location-wise stock balance, item expiration date warnings. Provide filters and export options.

Screens/Pages in Phase 2

This phase introduces numerous screens that extend the MVP:

- **Barcode/Label printing** – dedicated settings screen for configuring printers and generating labels (from module 2.4.6 and Tools 5.4 Printer Settings). Also **Report Templates** (5.5) for customised receipts and labels.
- **Inventory reconciliation** – **Inventory Reconciliation** page (2.4.4) with cycle count sessions and variance reports.
- **Stock adjustment reasons** – pages for **Invoice Adjustment Reason** and **GRN Adjustment Reason** masters (1.9–1.10).
- **Extra masters: Manufacturer** (1.5); **Banks & Branches** and **Ledger Accounts** (1.13–1.14);
- **Sales documents: Credit Note**, **Invoice Post Adjustment**, **Invoice Cancellation**, **Credit Invoice**, **Temp Invoice** (2.1.2–2.1.6).
- **Purchase documents: Purchase Return Note** (2.2.2); **GRN Post Adjustment** and **GRN Cancellation** (2.2.3–2.2.4).
- **Payments: Customer Receipt**, **Supplier Payment**, **Customer Payment Adjustment**, **Supplier Payment Adjustment**, **Customer Receipt Cancellation**, **Supplier Payment Cancellation** (2.3.1–2.3.6).
- **Advanced transfers: Stock Transfer Note** and **Stock Receive Note** (2.4.1–2.4.2) with transit status.
- **Reporting:** New fast/slow moving reports, ageing reports, item history and stock movement reports (6.1.14–6.1.22, 6.4, 6.5).
- **Settings:** Tools such as **User Type Authentication Mapping**, **Preferences**, **General Account Mapping** and **File Import** (5.6–5.9).

Phase 3 – Financial & Accounting Expansion

Goal

Implement the comprehensive financial and accounting modules required for a full POS/ERP system, including cash management, journals, ledgers, cheques and financial reporting. This phase corresponds to the **Financials & Accounts** epic in the charter and the `Cheques` and `Accounts` sections of the full scope.

Key features (functional)

Feature	Source & notes
Cash management	Record cash in/out transactions with reasons and track shifts/day-end cash counts. Full scope provides `Cash In Reason` and `Cash Out Reason` masters (1.11–1.12) and transactions for **Add Cash In/Out** (2.7.4–2.7.5).
General ledger & journals	Create general ledger accounts and journal entries; support cash/bank journal entries, bank reconciliation and closing periods.
Payments module integration	Process customer receipts and supplier payments and post them to the ledger, including adjustments and cancellations. Cheque management is also included.
Cheques management	Handle cheque deposits, deposits to fixed accounts, returned cheques and returned cheque payments.
Period closing routines	Day-end, shift-end, month-end and year-end closing with the ability to generate summaries and lock transactions.
Bank reconciliation	Reconcile bank statements with recorded transactions.
Financial reports	Provide trial balance, income statement, balance sheet, general ledger, cashflow and ageing reports.
PD/Deposit cheque details	Support post-dated (PD) cheque tracking and deposit details (6.6.1–6.6.2).

Technical implementation steps

1. **Design the general ledger:** Create tables for `accounts`, `journal_entries`, `journal_lines`, `bank_accounts` and `cash_drawers`. Define account types (asset, liability, equity, revenue, expense). Implement ledger posting triggers from sales, purchases and payments.
2. **Cash in/out:** Build forms for **Add Cash In** and **Add Cash Out** transactions. Link them to ledger accounts and track reasons using the corresponding master tables.
3. **Journal entries:** Provide UI for creating **Journal Entry** and **Cash/Bank Journal Entry** with double-entry validation. Users should be able to post adjustments and reclassifications.
4. **Bank reconciliation:** Develop a reconciliation interface where users import bank statements and match them against recorded transactions. Support marking transactions as reconciled and generating discrepancy reports.
5. **Cheques module:** Implement screens to record **Cheque Deposit**, **Cheque Deposit to Fixed Account**, **Returned Cheques** and **Returned Cheque Payments**. Link each cheque to a customer/supplier and update their outstanding balances.
6. **Period closing:** Build workflows for **Day End Close**, **Shift End Close**, **Month End Close** and **Close Financial Year**. When closing a period, summarise sales, purchases, payments and cash balances and lock further edits. Ensure that shift/day/month end reports (6.7.6–6.7.9) can be generated.
7. **Payments integration:** Extend payment modules from Phase 2 so that customer receipts and supplier payments post to the ledger. Handle payment adjustments and cancellations. Incorporate cheque payments and deposit tracking.

8. **Financial reporting:** Build report generators for **Trial Balance**, **Income Statement**, **Balance Sheet**, **General Ledger**, **Cash-flow**, **Accounts Receivable Aging** and **Accounts Payable Aging**. Reports should aggregate data across branches where permitted and support filtering by date range.
9. **Security & audit:** Apply strict RLS policies for financial tables and implement an audit trail of edits and approvals. Only authorised roles (e.g., accountant) may post journal entries or close periods.

Screens/Pages in Phase 3

- **Cheques:** `Cheque Deposit`, `Cheque Deposit to Fixed Account`, `Returned Cheques`, `Returned Cheque Payments` (2.5.1–2.5.4).
- **Accounts related:** `Journal Entry`, `Cash/Bank Journal Entry`, `Bank Reconciliation`, `Close Financial Year` (2.6.1–2.6.4).
- **Accounts entries:** `Day End Close`, `Shift End Close`, `Month End Close`, `Add Cash In`, `Add Cash Out` (2.7.1–2.7.5).
- **Reports:** `PD Cheque Details` and `PD Payment Details` (6.6); `Journal Listing`, `Trial Balance`, `Income Statement`, `Balance Sheet`, `General Ledger`, `Cash In/Out Report`, `Shift End Report`, `Day End Report`, `Month End Report` (6.7.1–6.7.9).
- **Masters: Banks & Branches**, **Ledger Accounts**, **Cash In Reason**, **Cash Out Reason** (1.11–1.14).

By the end of Phase 3, Flux becomes a fully integrated POS and accounting system capable of producing formal financial statements.

Phase 4 – Advanced Analytics & Optimisation

Goal

Complete the remaining full-scope requirements by delivering comprehensive reporting and optimisation tools, advanced dashboard widgets and system utilities. This phase focuses on analysis, forecasting and user experience polish.

Key features (functional)

Feature	Source & notes
Comprehensive stock reports	Implement the remaining stock reports: item list, stock quick find, supplier-wise stock summary/balance, location-wise stock balance, item history, stock transfer listing/find, item-wise stock movement and valuation (including as-at date).
Advanced sales and purchase reports	Provide invoice listings by date, user and sales rep; full sales summary by date/user; sales listing and GRN listing; purchase order listing; GRN summary and find GRN.
Customer & supplier analytics	Deliver outstanding invoice lists, customer area-wise summaries, accounts receivable aging, sales & receipt summary, customer statement; for suppliers, accounts payable aging, overdue GRNs, payment summary and supplier statement【821719869900778†L255-L287】.

Feature	Source & notes
PD & deposit cheque details	Finish PD cheque analytics, including PD cheque details and payment details (6.6.1–6.6.2).
Account-related reports	Finalise journal listing, trial balance, income statement, balance sheet, general ledger, cash in/out report and end-of-period reports (shift/day/month).
Dashboard widgets & visualisations	Build interactive dashboards combining multiple metrics: fast/slow moving items, non-moving items, top customers, branch comparisons, cashflow charts, sales versus purchases, etc. Provide date filters and export to CSV/PDF.
System tools & settings	Complete utilities: **Calculator** (5.1), user password change/recover (5.2–5.3), **Printer Settings** (5.4), **Report Templates** (5.5), **User Type Authentication Mapping** (5.6), **Preferences** (5.7), **General Account Mapping** (5.8) and **File Import** (5.9).
Optimisation & data import	Implement data import routines (CSV/XLSX) for items, customers and suppliers. Optimise queries and indexes for high-volume data. Introduce caching layers or read-replica strategies to scale across many branches.

Technical implementation steps

1. **Reporting engine:** Develop a generic reporting engine that executes parameterised SQL queries (against Supabase/Postgres) and renders tables, charts or exports. Use lazy loading and pagination for large datasets. Store report definitions and templates in the database for reuse.
2. **Dashboards & visualisations:** Integrate a charting library (e.g., Chart.js, Recharts) to build interactive dashboards. Allow users to configure widgets (choose metrics, date ranges, branches) and save personalised layouts.
3. **Complex queries & optimisation:** Write SQL views or materialised views for fast/slow moving items, non-moving items, ageing schedules and cashflow projections. Use indexes on `location_id`, `item_id`, `date` to improve performance. Implement caching (e.g., using SWR) to reduce network calls.
4. **Data import/export:** Provide CSV/XLSX import wizards for master data and transactions. Validate data client-side and server-side before insertion. Offer exports for all reports.
5. **System utilities:** Implement the remaining Tools pages (calculator, password management, printer settings, report templates, authentication mapping, preferences, account mapping, file import). Ensure each utility respects user roles and branch restrictions.
6. **User experience polish:** Add advanced offline analytics (e.g., offline dashboards), notification badges, dark mode, multi-language support and improved accessibility according to WCAG guidelines.
7. **Performance & scalability improvements:** Conduct load testing; optimise API calls; implement code splitting and lazy loading; compress assets; and refine caching strategies to maintain Lighthouse scores > 90.

Screens/Pages in Phase 4

- **Reports:** All remaining reports under sections 6.1–6.7 of the full scope (stock, sales, purchases, customers, suppliers, PD/Deposit cheques, account related).

- **Dashboards:** Customisable dashboard pages showing KPIs (fast moving items, top sellers, ageing). These may be new screens not explicitly listed in the full scope but derived from the data.
- **Tools:** Calculator (5.1), User Password Change (5.2), User Password Recover (5.3), Printer Settings (5.4), Report Templates (5.5), User Type Authentication Mapping (5.6), Preferences (5.7), General Account Mapping (5.8) and File Import (5.9).

Conclusion

This phased roadmap guides the solo developer from a minimal proof of concept to a comprehensive POS and ERP system. By adhering to the trimmed MVP definition and progressively incorporating deferred features, Flux can deliver value early while managing complexity. The later phases introduce operational refinements, robust financial modules and advanced analytics, culminating in the full scope envisioned in the project charter and scope documents.
