Flux POS & Inventory System – Phase 1 Sprint Plan

High-Level Sprint Plan

Sprint #	Name	Sprint Goal	Deliverables	Demo Scenario
1	**Roles & Core Masters**	Establish foundational data structures and role-based access so the POS/Inventory system has a secure baseline.	Core master tables (Branch, Role, User, Unit, Category, Item), role-based permissions policies, and UI forms for CRUD operations.	Show an administrator creating a new branch, defining cashier and manager roles, adding a user and assigning the role, then creating sample items with categories and units.
2	**Inventory & Batch Management**	Implement the inventory ledger with batch tracking and goods receipt so the system can accurately manage stock by lot and expiry.	Goods receipt (GRN) UI and APIs, batch creation logic, stock_moves and stock_balances tables, offline queue for GRNs, and expiry flagging on items.	Receive a shipment using the GRN form; show how batches are created with expiry dates; demonstrate that stock balances update and that expired batches are highlighted.
3	**Purchasing & Transfers**	Enable procurement and internal movements between branches to maintain stock levels across the organization.	Purchase order UI and RPCs, stock transfer documents and APIs, purchase return handling, and sync logic for these documents.	Create a purchase order for new stock, approve and send it; receive it into inventory; then transfer a quantity to another branch and observe balances update at both locations.
4	**POS Enhancements & Returns**	Deliver a functional point-of-sale with support for discounts, multiple payment types and return/refund flows.	POS screen with item lookup, batch selection, discount application, support for cash/credit/payment split, return/refund flows that adjust stock_moves appropriately, and offline sales queue.	Complete a sale with multiple payment methods, apply a promotional discount, then process a customer return; verify that the return updates stock and that managers can see a return document.
5	**Reporting & Final Polish**	Build essential reports and finalize Phase 1 functionality for a production-ready MVP.	Stock ledger, batch expiry, sales and purchase reports, role-based report access, final UI polish, documentation and user training materials.	Run a stock ledger report to show all stock_moves, filter to a specific item to verify balances; generate a batch expiry report highlighting soon-to-expire batches; run a sales summary report for the week.

Logical Flow: The plan begins with foundational data and permissions, proceeds to inventory and batch tracking, then to procurement and internal transfers, moves into POS functionality with returns, and concludes with reporting and polish. Each sprint builds on the previous one so that dependencies such as master data and stock ledger tables are in place before higher-level features are developed.

Sprint-by-Sprint Detail

Sprint 1 – Roles & Core Masters

Goal: Implement the core master data structures and role-based access control (RLS) so that subsequent modules can trust a consistent and secure foundation.

User Stories (P1):

Code	Title
P1-001	Set up Branch master and ability to add/edit branches
P1-002	Define roles (Admin, Manager, Cashier) and configure permissions
P1-003	Create User master and assign users to roles/branches
P1-004	Maintain Units of Measure and Item Categories
P1-005	Create Item master with category, unit and SKU fields
P1-006	Enforce row-level security so users only see data for their branch

Task Breakdown:

- **Frontend:**
- Build CRUD screens for Branch, Role, User, Unit, Category and Item masters.
- Include search/sort/pagination for ease of use.
- Add a basic navigation menu and login screen using existing Phase 0 framework.
- **Backend:**
- Design relational tables for branches, roles, users, units, categories and items.
- Implement RPCs or REST endpoints for CRUD operations with validation.
- Configure row-level security (RLS) policies so that a user with the "Cashier" role cannot read or write data outside their assigned branch.
- Seed initial roles (Admin, Manager, Cashier) and ensure permission inheritance.
- **Offline/Sync:**
- Set up synchronization scaffolding for master data: ensure masters created while offline queue properly and apply on next sync without duplication.
- **QA:**
- Unit tests for each CRUD RPC: create, update, delete and list operations.
- Integration tests verifying that a user logged in as "Cashier" cannot access data for another branch.
- Validate input constraints (e.g., unique SKU on Item master).

Dependencies: Completion of Phase 0 infrastructure (authentication, basic navigation, and database setup). No prior Phase 1 modules are required.

Acceptance Criteria:

- Users can successfully create, read, update and delete Branch, Role, User, Unit, Category and Item masters.
- Roles and permissions enforce access correctly (e.g., only Admins can create branches; cashiers cannot see other branches' masters).
- Item master requires valid unit and category; duplicate SKUs are rejected.
- Offline creation of masters syncs correctly without duplicating rows on retry.
- All new RPCs are covered by unit tests and pass continuous integration checks.

Sprint 2 – Inventory & Batch Management

Goal: Provide the inventory ledger and batch tracking with goods receipt processing so that stock levels and expiries are accurate across branches.

User Stories (P1):

Code	Title
P1-007	Create a batch record when stock arrives, including batch number and expiry date
P1-008	Capture Goods Receipt Note (GRN) and update stock_moves and stock_balances
P1-009	Maintain real-time stock_balances by item, batch and branch
P1-010	Flag expired batches in inventory and prevent sale
P1-011	Provide an offline queue for GRNs so receipts can be captured without connectivity

Task Breakdown:

- **Frontend:**
- Develop GRN form allowing users to select the supplier, date and items with quantities, batch numbers and expiry dates.
- Add inventory views: stock list by item/batch, with indicators for expired or soon-to-expire batches.
- Provide batch lookup in the Item master detail page.
- **Backend:**
- Define tables for stock_moves (transaction log) and stock_balances (current quantities by item/batch/branch).
- Implement RPCs to post GRN transactions: create batch entries, insert corresponding stock_move rows and update stock_balances.
- Add cron or trigger to automatically mark batches as expired once the expiry date passes.
- Enforce business rules: prevent GRN for unknown items or suppliers, and ensure quantities are positive.
- **Offline/Sync:**
- Build offline queue for GRN documents; ensure idempotent processing so that retries do not duplicate stock_moves.
- Implement last-write-wins conflict handling for GRN lines when the same batch is updated concurrently on different devices.
- **QA:**
- Test that posting a GRN increases stock_balances and creates the correct stock_move entries.
- Simulate expiry by setting a past expiry date and verify that the batch is flagged and cannot be selected at POS.
- Validate offline GRN creation and sync with network interruptions.

Dependencies: Master data (items, units, categories) and RLS must be in place. Supplier master may be assumed from Phase 0 or Sprint 1.

Acceptance Criteria:

- GRN posting creates batch, stock_move and stock_balance records with accurate quantities and dates.
- Expired batches are clearly indicated in the inventory view and cannot be used in POS transactions.
- Stock_balances reflect cumulative effects of GRNs; no duplication occurs during sync retries.
- Users can record GRNs offline and sync later; conflict resolution uses last-write-wins.
- Unit and integration tests confirm inventory calculations and expiry logic.

Sprint 3 – Purchasing & Transfers

Goal: Equip the system with procurement workflows and internal stock movements so that inventory can be replenished and balanced between branches.

User Stories (P1):

Code	Title
P1-012	Create and manage Purchase Orders (PO) for suppliers
P1-013	Receive stock against a PO and reconcile with the order
P1-014	Initiate inter-branch stock transfers and update stock_moves
P1-015	Handle purchase returns and adjust stock_balances accordingly
P1-016	Apply RLS policies to ensure branch-specific PO and transfer visibility

Task Breakdown:

- **Frontend:**
- Build PO listing and detail screens with status (Draft, Approved, Received).
- Add a transfer request screen where users can select source and target branches, items and quantities.
- Provide UI to process purchase returns, selecting the original GRN line and entering quantities being returned.
- **Backend:**
- Create tables for purchase_order and purchase_order_line with foreign keys to supplier and item masters.
- Implement RPCs for PO creation, approval and receiving; update stock_moves and stock_balances on receipt.
- Develop stock transfer RPCs that decrement stock from the source branch and increment stock in the target branch.
- Add purchase return logic that reverses the original stock_move and adjusts stock_balances.
- **Offline/Sync:**
- Support offline capture of POs and transfers with queues similar to GRN.
- Ensure idempotent updates and maintain PO status transitions across devices.
- **QA:**
- Create a PO, receive goods and verify that receiving updates inventory and closes the PO.
- Transfer items between branches and confirm that balances update accordingly for both branches.
- Process a purchase return and ensure that returned quantities are deducted from inventory and returned to supplier.
- Verify branch-based RLS: a user from Branch A cannot access Branch B’s POs or transfers.

Dependencies: Sprint 1 masters and Sprint 2 inventory ledger must be functional. Supplier master should be populated.

Acceptance Criteria:

- Users can create, approve and receive POs; receiving updates stock_balances and stock_moves.
- Transfer transactions correctly adjust inventory in both source and target branches, with proper audit trail.
- Purchase returns reverse stock movements and close the return transaction with reference to the original GRN/PO.
- Offline capture and syncing of POs and transfers works without duplication; last-write-wins resolves conflicts.
- All actions respect RLS policies and are covered by unit/integration tests.

Sprint 4 – POS Enhancements & Returns

Goal: Deliver a robust point-of-sale module with support for discounts, multiple payment types, returns and stock adjustments, enabling real-time sales operations.

User Stories (P1):

Code	Title
P1-017	Perform sales transactions at POS with item scan and quantity entry
P1-018	Allow discounts (percentage or amount) on individual line items or entire ticket
P1-019	Accept multiple payment types (cash, card, voucher) per transaction
P1-020	Process sales returns and refunds, adjusting stock_moves and balances
P1-021	Prevent negative inventory at POS and warn on insufficient stock
P1-022	Select specific batches at POS and prevent sale of expired batches
P1-023	Maintain offline sales queue for later sync

Task Breakdown:

- **Frontend:**
- Develop POS screen with fast item lookup (barcode/SKU search), quantity and price entry, discount fields and payment options.
- Add ability to search and select a batch when multiple batches exist for an item; highlight expiry dates.
- Build return UI: search for original receipt, select items to return and specify quantities; calculate refund amounts.
- Show warnings when stock is insufficient or the selected batch is expired.
- **Backend:**
- Implement RPCs for sales posting: create stock_move entries that decrement stock_balances and generate sale documents.
- Support multiple payment types with proper tender balancing and change calculation.
- Add discount logic and rules; ensure discounts are recorded and reflected in reports.
- Handle sales returns by reversing stock_moves and creating a credit/refund transaction.
- Enforce prevention of negative stock and blocking of expired batches at the server.
- **Offline/Sync:**
- Maintain an offline sales queue; ensure sequential numbering and idempotent sync when connectivity resumes.
- Apply last-write-wins conflict resolution if the same receipt is edited on different devices (e.g., price adjustment vs discount).
- **QA:**
- Execute end-to-end POS transactions, including multiple payment splits and discounts, and verify inventory updates and receipts.
- Attempt to sell quantities exceeding available stock and confirm that the system blocks or warns accordingly.
- Perform returns and confirm that stock and financials are updated; test both full and partial returns.
- Validate offline sales capture and later synchronization under network interruptions.

Dependencies: Sprints 1–3 must be complete, especially inventory balances and batch logic. Payment integrations (if any) must be configured.

Acceptance Criteria:

- POS allows rapid entry and completion of sales with multiple payment methods and discounts.
- The system prevents selection of expired batches and warns when stock would go negative.
- Sales returns create correct reversal entries and adjust stock_balances.
- Offline sales sync without duplication or loss of receipt numbers.
- All POS functions respect RLS and pass functional and integration tests.

Sprint 5 – Reporting & Final Polish

Goal: Produce essential operational reports and finalize Phase 1 deliverables to deliver a production-ready MVP.

User Stories (P1):

Code	Title
P1-024	Generate stock ledger report showing all stock_moves per item/batch
P1-025	Produce batch expiry report listing batches nearing or past expiry
P1-026	Create sales summary report with filters by date, branch and cashier
P1-027	Build purchase report summarizing POs and receipts
P1-028	Provide role-based access to reports
P1-029	Perform final UI polish and user training materials

Task Breakdown:

- **Frontend:**
- Develop report selection screens with date and branch filters and export options (CSV/PDF).
- Implement drill-down capabilities from summary to transaction details (e.g., from sales summary to individual receipts).

- Apply UI polish: consistent colors, responsive layouts and improved navigation.
- Draft user training guides embedded in the help menu or as downloadable PDFs.
- **Backend:**
- Build SQL queries or views to support each report, using stock_move, stock_balance, sales and purchase tables.
- Implement RPCs to generate reports with pagination and filtering; ensure they respect RLS (e.g., a manager sees their branch's data only).
- Optimize queries for performance on large datasets; consider indexes on date and branch columns.
- **Offline/Sync:**
- Ensure that reports can still be generated on offline devices using locally cached data, or gracefully warn users when fresh data is required.
- **QA:**
- Validate each report for accuracy by cross-checking totals with underlying transactions.
- Test role-based access: only authorized users can view specific reports.
- Execute performance tests on reports with large datasets to verify acceptable load times.

Dependencies: Completion of all prior sprints with clean and consistent data.

Acceptance Criteria:

- Stock ledger, batch expiry, sales and purchase reports return accurate, filtered data.
- Reports enforce branch-level RLS and support export to CSV/PDF.
- UI is polished and consistent; training materials are available.
- All regressions from previous sprints are resolved; integration tests pass.

Phase 1 Testing Plan

The testing strategy for Phase 1 focuses on validating inventory correctness, batch handling and synchronization stability.

1. **Inventory Logic**
2. Verify that each GRN increases stock_balances and creates corresponding stock_move entries. After a sale, confirm that quantities decrease accordingly. Cross-check with the stock ledger report.
3. Create multiple GRNs and sales for the same item and ensure that cumulative balances remain consistent. Perform audits at the branch and item levels.
4. Test negative cases: posting a GRN with an invalid item or zero quantity should throw a validation error.

5. **Batch Tracking**

6. Post GRNs with different expiry dates and confirm that batches are listed with expiry flags. Use a mocked current date to trigger expiry and ensure that expired batches are excluded from POS selection and appear in the batch expiry report.
7. Sell items from a specific batch and verify that stock_moves record the batch_id. When the batch runs out, the POS should automatically prompt for the next available batch.
8. Attempt to sell from an expired batch: the system should prevent the sale and display a warning.

9. **Sync Conflicts**

10. Simulate offline operation by disconnecting a device, posting GRN and sales transactions, then reconnecting. Confirm that transactions sync without duplication.
11. Create conflicting updates (e.g., two users editing the same GRN line while offline). Ensure that the last-write-wins policy from the scope document resolves the conflict and leaves the system in a consistent state. Capture audit logs for visibility.
12. Retry syncing after a partial failure and verify that no duplicate stock_moves or stock_balances entries are created ("No data duplication on sync retries").

13. **Role-Level Security (RLS)**

14. Test that each API and report enforces branch-level data isolation. Use accounts with different roles (Admin, Manager, Cashier) and confirm that data visibility aligns with permissions.
15. Attempt unauthorized actions (e.g., a cashier trying to approve a PO) and verify that access is denied.

16. **Performance and Usability**

17. Run performance tests on heavy reports and POS operations with thousands of transactions to ensure acceptable response times.
18. Conduct usability tests with end users to refine navigation, workflow and error messaging.

Definition of Done for Phase 1

The MVP for Phase 1 is considered complete when the following conditions are met:

1. **Functional Completeness** – All P1 user stories and acceptance criteria listed above are implemented. Out-of-scope features (e.g., accounting integration, barcode printing) are deferred to Phase 2 and are not partially implemented.

2. **Quality Assurance** – Unit, integration and end-to-end tests pass. Inventory logic produces correct stock_balances and stock_moves. Batch tracking and expiry functions work correctly. Sync logic is robust: retries do not duplicate data, and last-write-wins resolves conflicts deterministically.

3. **Security & RLS Policies** – Row-level security prevents users from accessing or modifying data outside their branch. All APIs and reports respect role permissions. Attempts to bypass RLS (e.g., by modifying request payloads) are rejected.

4. **Offline & Sync** – All document types (masters, GRNs, POs, transfers, sales, returns) can be captured offline and synced later. Sync operations handle partial failures gracefully and never create duplicate entries on retry. Conflict resolution is implemented using last-write-wins.

5. **User Experience** – The user interface is cohesive and polished. Key workflows (masters, inventory, purchasing, POS, reporting) are intuitive and well documented. User training materials are available. There are no blocker bugs, and non-critical bugs are logged for Phase 2.

6. **Documentation** – Technical documentation (data models, API specifications, RLS policies) and user documentation are updated. Deployment scripts and environment configuration are ready for production rollout.

This sprint plan provides a roadmap for delivering a trimmed MVP in Phase 1. It follows a logical dependency order, assigns specific user stories and tasks to each sprint, outlines a comprehensive testing approach, and sets a clear definition of done to ensure readiness for release.