Product Requirements Document (PRD) — setLedger

Prepared for: your team / stakeholders
Date: 2025-10-03 (Asia/Kolkata)

Great — I've expanded the PRD into a full, implementation-ready specification for **setLedger**, incorporating every change you requested and going deep on each segment: features, improvements, fallbacks, tech stack, data & security, AI, modular architecture, step-by-step implementation guidance, MVP backlog with acceptance criteria, and operational details. I did **not** omit anything you or I suggested earlier.

---

1) Executive summary (short)

setLedger is a modular, open-source financial & commerce OS targeting every scale of business — from single-shop micro-retailers to large enterprises — offered with a truly free core. Inspired by Odoo but redesigned to fix common pain points, setLedger adds: product QR **generation**, robust multi-tenant identity (orgID+memberID userIDs), strong authentication (username/password + TOTP), offline-capable POS with local backups, a reliable modular architecture (loose coupling, strong contracts), first-class error handling & fallbacks, and a powerful hybrid AI finance engine (local + centralized) with local storage backup options.

High-level mission: *Free core, safe customizations, low implementation cost, enterprise-grade reliability, strong AI for financial decision support, and privacy-preserving defaults.*

---

2) Objectives & success metrics

Primary objectives

- Deliver an MVP (billing with QR, POS, inventory, accounting/GST, dashboards, AI stock/pricing assistant).
- Make onboarding frictionless: create first invoice < 2 hours for micro-retailer.
- Ensure secure multi-tenant operation with centralized signup on `app.setledger.com` and unique userIDs.
- Ensure offline capabilities + encrypted local backups for continuity.

Key 12-month metrics

- 10k org signups (self-host or free SaaS).
- 40% of signups issue invoices within 24 hours.
- 50% retention after 90 days.

- AI forecast MAPE < 20% for customers with ≥6 months data (pilot).
- NPS: free users ≥ +30; paid users ≥ +60.

---

3) Target users (personas)

- **Micro-retailer** — wants simple POS, product QR & GST invoice.
- **SME distributor** — multi-warehouse, purchase orders, GST filing helpers.
- **E-commerce seller** — sync inventory across channels, auto-invoicing.
- **Accountant / CA** — multi-client ledgers, reconciliations, audit trails.
- **Enterprise ops** — multi-company consolidation, governance & ACL.

---

4) High-level architecture & modularity principle

Principles

- **Modules = services with well-defined contracts** (REST/gRPC + events). Examples: Auth, Org, Product, Inventory, Sales, POS, Accounting, AI, Billing, Reporting, Integrations, Backup.
- **Loose coupling + strong data contracts** — versioned APIs, schema validation, and semantic versioning for modules.
- **Single platform domain**: `app.setledger.com` hosts org onboarding; orgs can optionally map custom domains.
- **Multi-tenant model**: hybrid — shared schema for service-level resources + per-tenant data partitioning (tenant_id column + row-level security).
- **Module integrity**: each module has its own DB schema namespace (or microservice DB) but exposes canonical DTOs and event types so modules can operate independently or be swapped.

Benefits

- Independent deployment & rollback
- Fine-grained scaling
- Easier testing/sandboxing
- Clear upgrade paths

---

5) Identity, Authentication & Authorization (critical)

Requirements

- Username + password (hashed with Argon2/Bcrypt) + TOTP (Google/Microsoft Authenticator standard).
- Optional SSO (OpenID Connect) for enterprises later.
- Admin invites & org onboarding from main domain.
- Unique userID format composed of organisationID & memberID.

UserID scheme (recommended)

- `ORG-<8hex>-U-<6dig>` e.g. `ORG-1a2b3c4d-U-000123`
    - `ORG-<8hex>` = short org id (derived from UUID v4 truncated)
    - `U-<6dig>` = zero-padded member sequence within org
- Store user_uuid (UUIDv4) as canonical primary key; `userID` string used for human/reporting and cross-system references.

Auth flows

- Sign-up (org admin) on `app.setledger.com` → generates `orgID` & admin user. Admin invites members via email link -> members create username/password + enable TOTP.
- TOTP onboarding: generate secret (store encrypted), show QR for authenticator apps, show printable backup codes once.
- Recovery: email-based recovery + one-time backup codes (no TOTP secret re-send); optional org admin override with audit trail.

Security safeguards

- Rate limiting, account lockout on repeated failed attempts, password strength enforcement, device/session management, session revocation, 2FA mandatory for admin role.
- Store TOTP secrets encrypted (AES-256) with KMS or HSM for hosted tenants; for self-hosted, instruct on secure key storage.

Libraries / tech

- Node: `otplib`, `speakeasy` or Python: `pyotp`. Use library best practices.

---

6) Billing / Invoicing & Product QR generation

Core features

- Generate GST-compliant invoices with line items, taxes, discount rules, HSN/SAC, invoice-level QR and **product-level QR codes** (for label printing).
- Two QR types:
    - **Product QR**: encodes productID, SKU, HSN, batch/expiry (optional), default price and link to product page (short URL on setLedger). Format: `setledger://p/<orgID>/<productUUID>` and fallback `https://app.setledger.com/p/<orgSlug>/<productSlug>`.
    - **Invoice QR**: encodes invoice id, total, payment metadata (payment link), and GST summary (if required).
- QR generation server-side and client-side (to support offline label printing): use JS/TS QR libs or server libs.

Product QR generation features

- Bulk QR generation for product catalog (PNG / SVG / PDF label sheet).
- QR layout templates for thermal printers (58mm, 80mm) and label sheets (A4 sticker layout).
- Option to embed dynamic content (price & stock snapshot) — but product QR remains canonical identifier (server resolves latest metadata).

Scanning & fallbacks

- Scanner priorities: QR → barcode (EAN/UPC) → manual SKU entry.
- Client-side scanning lib options: `zxing-js`, `jsQR`, `QuaggaJS` for browsers; native libs for mobile.
- If product lookup fails (server unreachable): client uses cached product metadata (IndexedDB). If no cache, prompt manual entry and create temporary transaction object to sync later.

Acceptance

- Can generate printable QR labels in batch for 1k SKUs within a minute (backend).
- Invoices show QR with payment link and GST summary; QR scans open payment page or adds product to invoice when product QR scanned.

---

7) Point of Sale (POS) — offline-first & resilient

PWA-based POS

- Works in browser & mobile; supports hardware barcode scanner, receipt printer, cash drawer.

- **Offline-first**: local persistence with IndexedDB, service workers for asset caching, background sync for queued transactions.
- **Conflict resolution**: optimistic writes + server reconciliation. Each POS has a `device_id` and local sequence numbers for transactions.
- **Offline fallbacks**:
    - Continue accepting sales, print receipts locally; on reconnect, sync transactions; reconcile stock with server via CRDT-style merge with last-write-wins but prefer server's reserved inventory on conflict and mark discrepancies for staff.
    - If reconciliation fails, flag transactions for manual review.

POS features

- Fast product search, scanning, discounts, returns, shift management, cash reporting.
- Shift-close includes signature, shift totals, and optional end-of-day snapshot.

Acceptance

- Add-to-cart latency < 200ms normally; fully functional offline mode for typical transaction volumes.

---

8) Inventory & Stock Management

Core features

- Multi-warehouse, locations, stock moves, reservations.
- Serial/batch management, expiry dates.
- Valuation methods (FIFO/LIFO/Avg).
- Stock counts / cycle counts, adjustments.

AI-powered inventory

- Reorder points, EOQ, lead-time calculations, safety stock suggestion.
- Slow / dead stock detection; markdown suggestions.

Fallbacks & error handling

- If AI or connector down: preserve manual reorder rules & show informational banner.
- Provide "urgent reorder" manual override with offline PO creation and later sync.

Acceptance

- Stock reservation on order creation; conflicting reservations show clear conflict resolution UI.

---

9) Accounting & GST compliance (India-first, extensible)

Core features

- Chart of accounts template per country, journals, ledger, bank reconciliation.
- GST fields on invoices (IGST/CGST/SGST), HSN/SAC per line, GST returns export (CSV/JSON) for filing tools.
- Audit trails: immutable journal entries, ledger snapshots.

E-invoicing & filing

- Support generating e-invoice payloads (where required) and export formats compatible with tax portals.

Fallbacks

- If filing connectors fail, generate validated offline exports (CSV/JSON/XML) so accountant can upload manually.

Acceptance

- Complete an invoice → journal entry auto-created and reconciled when payment arrives.

---

10) Pricing Engine & Sales Techniques

Features

- Pricing strategies: cost-plus, target-margin, dynamic pricing (time-window / demand).
- Tiered pricing, customer-group pricing, promo engine, coupon codes.
- A/B pricing module (control & variant groups, basic conversion tracking).
- Rule-builder UI (low-code): "If X and Y then set price = formula".

Fallbacks

- If pricing engine unavailable, the system uses last cached approved price list.

Acceptance

- Admin can create pricing rule and see preview of impact on sample SKUs before enabling.

---

11) AI Finance Engine (design & fallbacks)

Goals

- Demand forecasting (per SKU), anomaly detection (fraud/dup orders), price-sensitivity suggestions, working-capital recommendations, cash flow forecasting.

Architecture

- **Hybrid design**:
    - **Local models**: lightweight time-series models (Prophet/ARIMA or small PyTorch models) for immediate inference on-prem / in browser (WebAssembly) for offline support.
    - **Centralized training**: heavier models trained in cloud/hosted offering to improve accuracy (multi-tenant aggregated learning with differential privacy — opt-in).
    - **Model serving**: model-as-a-service microservice with versioning; local fallback endpoint returns rule-based heuristics.
- **Explainability**: show drivers for predictions (seasonality, lead-time, promotions) using SHAP-style explanations.

Fallbacks

- If model inference unavailable, fall back to heuristics:
    - moving-average + safety buffer
    - last-X months average normalized for seasonality

Data & privacy

- Default: local-only training for each org. Offer opt-in aggregated training that anonymizes data (hashing, k-anonymity) for cross-tenant improvements.

Tech

- Model registry (MLflow), training pipelines (Airflow), PyTorch / scikit-learn, on-device inference via ONNX/TF Lite where needed.

Acceptance

- Provide reorder recommendation with confidence bands; allow manual override and log changes.

---

12) Dashboard & Analytics

Dashboards

- Role-based default dashboards: Owner (top-line), Accountant (P&L, Balance sheet), Store (stock velocity).
- Drill-down capability to transactions, SKU, customer, period.
- Scheduled reports (email or CSV).

Analytics tech

- Use OLAP/analytics store (ClickHouse or PostgreSQL + materialized views) for fast KPIs.
- Ad-hoc query UI; export to CSV/Excel.

Fallback

- If analytics service is slow, show cached snapshot + "last updated" timestamp.

---

13) Integrations & Marketplace

Integrations

- Payment gateways (developer/free-tier accounts), shipping partners, eCommerce (Shopify/Amazon), bank statement import, GST e-invoicing portals.

Marketplace

- Certified connectors, free & paid plugins, extension guidelines, sandbox testing before listing.

Use free APIs where possible

- Scanning libs: `zxing-js`, `jsQR`, `QuaggaJS` (open-source).
- QR generation: `qrcode` (node), `segno` (python).
- Currency rates: open/free services (e.g., exchangerate.host or similar free APIs).
- Email: SMTP/Gmail or Sendinblue free tiers / SMTP relay.
- Maps/geo: OpenStreetMap.
- Vector DBs: FAISS (local) or Milvus (open-source).

(Implementation should pick exact providers after discovery; preference for OSS & free tiers for core features.)

---

14) Data model (high-level)

Core entities: Company(Org), User, Role, Product, SKUVariant, PriceList, InventoryLocation, StockLot, StockMove, PurchaseOrder, SalesOrder, Invoice, Payment, BankStatement, FiscalPeriod, GSTDocument, ForecastModel, BackupSnapshot, Connector, EventLog.

Store tenant_id on each row for multi-tenancy; use UUID primary keys; index on org_id, product_sku, created_at.

(If you want, I’ll produce a full ERD SVG next.)

---

15) Non-functional requirements

- **Scalability:** service partitioning & horizontal autoscale (K8s), DB sharding strategies for future scale.
- **Performance:** POS latency <200ms, dashboard queries <1s for top KPIs.
- **Availability:** 99.9% SLA for hosted tier; core self-hosted with clear HA guidance.
- **Security:** encryption-in-transit (TLS), encryption-at-rest, RBAC, audit logs, 2FA.
- **Localization:** date/currency/tax templates per country; India/GST ready by default.
- **Observability:** centralized logging, metrics, distributed tracing, error tracking.

---

16) Error handling & robust fallbacks (detailed)

Principles

- Fail-safe first: when an optional service fails, app remains usable (with reduced functionality).
- Clear user feedback (error banners, suggested actions).
- Automatic retries with exponential backoff for transient errors.
- Circuit breaker for flaky external services.

Module-specific fallbacks

- **Auth**: fallback to token refresh; when TOTP provider malfunction (really we’re self-hosting TOTP, so no external dependency).
- **POS offline**: queue transactions locally; show "sync pending" and self-heal conflicts.
- **QR scanning**: when camera permission denied, allow manual SKU entry or use built-in barcode scanner fallback.
- **AI engine**: fallback to heuristics/local simple models.
- **Integrations**: if bank feed fails, allow manual CSV upload and schedule connector retry.

Error handling patterns

- Structured errors with codes + human message + remediation path.
- Logging with correlation IDs to trace user flows across services.
- Alerting thresholds: error rate spikes, sync failure rate, queue length.

---

17) Backups & local storage (mandatory)

Local backup strategy

- POS/local-app: persist to IndexedDB and use user-initiated encrypted exports (JSON/zip). Provide "Export now" and scheduled auto-export to local USB/NAS.
- Server-side backup: scheduled, encrypted DB dumps (logical + physical), WAL archiving; incremental snapshots and retention policy.

Encryption & integrity

- AES-256 encryption for backups; sign all backups (HMAC) for integrity.
- Provide recovery CLI with dry-run mode and schema validation.

Restore

- Atomic restore pipeline: verify checksum → restore to staging → run sanity checks → switch.

Backup fallback

- If remote backup provider unavailable, fallback to local NAS or customer-provided SFTP endpoint.

---

18) Tech stack (recommended per layer)

Frontend

- React (Typescript) + PWA + Tailwind for UI; use React Query or SWR for data sync.
- POS: PWA with service workers + IndexedDB.

Backend

- Option A (Node): NestJS + TypeORM / Prisma
- Option B (Python): FastAPI/Django + SQLAlchemy
  Choose based on team familiarity. Both work well.

DB & storage

- PostgreSQL primary (ACID), Redis cache, S3-compatible object storage (MinIO for self-hosted).

AI/ML

- PyTorch / scikit-learn / Prophet; MLflow for model registry; ONNX for model portability.

Search & analytics

- Elasticsearch or ClickHouse for analytics; or PostgreSQL materialized views with read replicas.

Queue & events

- RabbitMQ / Kafka (Kafka for heavy event throughput or RabbitMQ for simpler needs).

CI/CD & infra

- GitHub Actions / GitLab CI, Docker, Helm charts, Kubernetes (EKS/GKE/AKS or k3s for small infra).
- Monitoring: Prometheus + Grafana; Error tracking: Sentry.

Devops & secrets

- HashiCorp Vault or cloud KMS (AWS/GCP) for secret management.

---

19) Testing & QA

- Unit tests (coverage target 80% for core modules).
- Integration tests for service contracts and API compatibility.
- E2E tests (Cypress) for core flows – onboarding, POS sale, invoice generation.
- Load testing for POS and API endpoints.
- Regression suite for customizations & low-code rules.
- Security testing (SAST, DAST) and regular vulnerability scans.

---

20) Roadmap & detailed implementation steps (what to focus on — stepwise)

I'll break this into **phases** and **sprints** with prioritized focus. Don't worry — I included everything you asked.

Phase 0 — Discovery & infra (2 weeks)

1. Finalize data model & API contracts for core modules (Auth, Org, Product, Inventory, Sales, Invoice).
2. Choose backend stack (Node or Python). Draft deployment manifests (Docker-compose + Helm stub).
3. Implement core CI/CD skeleton, code style, PR templates, contributor guide.
4. Create org/userID scheme & auth flow spec.

Focus: multi-tenant model & auth — this underpins all other features.

Phase 1 — Core MVP (8–12 weeks)

Sprints (2 weeks each) — deliverables:

Sprint 1–2 (Auth + Org + Product)

- Auth service: username/password, TOTP onboarding UI, recovery & audit logs.
- Org service: signup flow on `app.setledger.com`, org admin invite, userID generation.
- Product service: CRUD, product QR generation endpoint (PNG/SVG), bulk QR export.

Sprint 3–4 (Invoicing + POS basics)

- Invoice module: create invoice, PDF generator, invoice QR (payment metadata), GST fields.
- POS PWA skeleton: offline storage (IndexedDB), product lookup via QR/barcode, local QR generation fallback.

Sprint 5–6 (Inventory + Accounting basics)

- Inventory moves, stock reservations, basic ledger & journal entries for sales invoices.
- Bank statement manual import.

Sprint 7 (Backups & Monitoring)

- Local/Server backup pipelines; basic observability.

Focus: make core transactions (product → sale → invoice → journal entry) work reliably offline & online.

Phase 2 — AI & Pricing Engine (8 weeks)

- Implement local forecasting (Prophet/ARIMA) + rule-based fallback.
- Model-serving microservice + MLflow.
- Pricing rule-builder UI, A/B pricing experiments.

Focus: Explainability & safe default (rule backup).

Phase 3 — Integrations, Marketplace & Scale (8–12 weeks)

- Payment gateways, shipping connectors, eCommerce sync.
- Marketplace and plugin certification.
- Hosted SaaS pipeline, multi-region deployment, hardened security & compliance.

Focus: partner connectors & hosted offering.

---

21) MVP Backlog (top user stories + acceptance criteria)

I'll show the top 20 to get started — you can ask me to expand into a CSV/backlog board.

1. **As an org-admin I can sign up on `app.setledger.com` so that I can create my company account.**

- AC: Create org, generate orgID, admin user created, admin receives invite email.

2. **As admin I can invite members; each member gets a unique userID.**
    - AC: Invite link -> member registers -> userID format `ORG-xxxx-U-000nnn` created; stored user_uuid.
3. **As a user I can enable TOTP; I see QR & backup codes.**
    - AC: TOTP secret stored encrypted; login with TOTP succeeds; backup codes validated.
4. **As a product manager I can create products & bulk-generate product QR labels.**
    - AC: Generate downloadable PDF/A4 label sheet with QR and SKU; QR decodes to canonical productID.
5. **As cashier I can scan a product QR and add it to POS cart (offline and online).**
    - AC: Works offline with cached product metadata; syncs on reconnect.
6. **As store I can take a sale; POS prints receipt & creates invoice with invoice QR.**
    - AC: Sale recorded locally, synced to server, invoice PDF with QR available.
7. **As accountant I can see auto-created journal entries for invoices.**
    - AC: Invoice → journal entry mapping created, viewable and exportable.
8. **As inventory manager I can transfer stock between warehouses and perform cycle counts.**
    - AC: Stock moves created and stock levels updated accurately.
9. **As owner I can view P&L dashboard and drill down to transactions.**
    - AC: Dashboard shows revenue, profit, top SKUs; drill-down works.
10. **As admin I can download encrypted backup & restore in staging.**
    - AC: Restore validates checksum and passes sanity checks.

(… + 10 more stories for pricing, AI recommendations, connectors — I can list on request.)

---

22) Acceptance criteria & KPIs per module (brief)

- **Auth**: 2FA enabled for admins; login attempts tracked; lockout after 5 failed attempts.
- **POS**: Offline transactions preserved with 100% data integrity on sync in 95% of cases.
- **Invoice**: GST fields validated; invoice QR opens payment page.
- **AI**: For pilot stores, MAPE < 20% after 6 months of data.

---

23) Monitoring & operations

- Logs: structured (JSON) with correlation IDs.
- Error tracking: Sentry (or open-source alternative).
- Metrics: Prometheus + Grafana dashboards (uptime, error rate, sync queue length).
- Alerts: Slack/Email for critical failures (sync backlog > threshold).

---

24) Compliance, privacy & policies

- Default retention & data export tools.
- GDPR-like export & delete flows.
- Audit trails and immutable journals for tax audits.

---

25) Team & role recommendations (initial)

- Product manager (1)
- Lead engineer (1)
- Backend engineers (2–3)
- Frontend engineers (2)
- ML engineer (1)
- DevOps (1)
- QA (1)
- UX (1)
- Tax/legal consultant (contract)

---

26) Risks & mitigations (short)

- **Complex customization pain** → mitigation: low-code rule builder, test harness, sandbox mode.
- **Data loss** → mitigation: encrypted backups, local device persistence, recovery playbooks.
- **AI cold-start** → mitigation: heuristics + optional federated learning opt-in.

---

27) Deliverables I can produce next (pick one or I’ll pick highest ROI)

I can immediately generate any of these — say which you want **first** (or I’ll start with the MVP backlog in full detail):

1. Full prioritized **MVP backlog** (Jira-style stories + AC + estimates).
2. Technical **architecture diagram** (K8s services, data flows).
3. **ERD** (Postgres schemas + fields for core tables).
4. **Onboarding wizard wireframes** (screens for admin onboarding + POS flow).
5. Implementation-ready **Helm + Docker-Compose** quickstart manifests.

---

28) Actionable step-by-step focus (what to do now — developer sprint plan)

Week 0 (pre-sprint)

- Approve PRD & choose stack (Node vs Python). Decide hosted vs self-hosted pilot.

Sprint A (2 weeks)

- Implement Auth + Org signup + userID generation + TOTP UI. Create CI skeleton.

Sprint B (2 weeks)

- Product service + product QR generation + bulk export.

Sprint C (2 weeks)

- Invoice creation + PDF/QR invoice + journaling.

Sprint D (2 weeks)

- POS PWA core (offline storage & sync) + scanning.

Sprint E (2 weeks)

- Inventory reservation + reconciliations and backups.

Sprint F (2 weeks)

- Basic AI forecasting local model + UI for reorder suggestions.

Deliver after 3 months: internal Beta for 3 pilot merchants.

---

29) Final notes & constraints

- I prioritized **safety, reliability, and offline continuity** in all designs as requested.
- The PRD intentionally favors **open-source / free-tier tools** for core capabilities; specific vendors for hosted connectors/payment providers should be selected during discovery.
- I included both **technical** and **operational** guidance so a small lean team can start building immediately.