

## System Architecture

### 1. Architecture Overview

MEDLOCUS is a modular, cloud-friendly, microservices-style platform that unifies **Inventory, Sales & Billing, Customer/Prescription Management, Analytics, and Compliance** through a secure API layer, realtime sync, and a managed relational data store.

### 2. High-Level Components & Responsibilities

- **User Interface (UI)**
  - Web client: Next.js (React + TypeScript) — point-of-sale counters, manager dashboards.
  - Mobile client (optional): React Native / Flutter for on-the-go access.
  - Responsibilities: UX, form validation, offline caching for POS, charts/dashboards.
- **API Gateway / Edge**
  - Single entry point for client traffic.
  - Responsibilities: authentication enforcement, rate limiting, request routing, TLS termination, simple request validation.
- **Backend Services (logical services)**
  - **Inventory Service**
    - Handles stock levels, batch numbers, expiry, stock reconciliation, supplier orders.
    - Ensures consistent stock updates (idempotent APIs, optimistic locking or DB transactions).
  - **Sales & Billing Service**
    - Generates invoices (GST-ready), processes transactions, supports POS operations and returns.
  - **Customer & Prescription Service**
    - Manages customer profiles, secure prescription records, refill schedules and reminders.

- **Notification Service**
  - SMS / Email queueing (Twilio/SendGrid or regional provider) and scheduling of reminders & alerts.
- **Analytics & Reporting Service**
  - Aggregates data for dashboards, demand forecasting and exportable compliance reports.
- **Auth & User Management Service**
  - Issue/validate tokens (JWT), manage roles (pharmacist, manager, auditor), and session lifecycle.
- **Integration / Sync Service**
  - Handles supplier portals, accounting exports, GST integrations, and multi-branch synchronization.
- **Data Layer**
  - **Primary DB:** Managed PostgreSQL (ACID for inventory & billing).
  - **Cache / Fast store:** Redis for hot lookups, session store, distributed locks.
  - **Search / Analytics store (optional):** Elasticsearch or analytical DB for fast reporting, full-text product search.
  - **Object Storage:** S3-compatible for backups, invoice PDFs, scanned prescriptions.
  - **Audit Log Store:** Immutable append-only store (S3 + metadata in DB or WORM storage) for compliance.
- **Message Bus / Eventing**
  - Kafka / RabbitMQ / AWS SNS+SQS for decoupled events: stock-changed, sale-completed, expiry-alert, backup triggers, cross-service communication.
- **Observability**
  - Centralized logging (ELK/Cloud provider), metrics (Prometheus), dashboards/alerts (Grafana), error tracking (Sentry/Datadog).
- **CI/CD & Infra**
  - Docker images, automated pipelines (GitHub Actions), IaC (Terraform), container orchestration (ECS/EKS or managed Kubernetes).

### 3. Data Flow (typical scenarios)

#### 1. Sale at POS

- UI → API Gateway → Sales Service → Transaction persisted in PostgreSQL within transaction.
- Sales Service emits sale.completed event to message bus.
- Inventory Service consumes event → decrement stock (with distributed lock/transaction).
- Notification Service consumes event → send customer receipt and update analytics.

#### 2. Stock Receipt

- Inventory: supplier invoice uploaded → Inventory Service validates batch & expiry → update DB → emit stock.updated → Analytics/Reporting update.

#### 3. Expiry / Low-Stock Alert

- Cron / Event processor queries DB or consumes inventory events → if threshold reached emit alert.lowstock/alert.expiry → Notification Service schedules message.

#### 4. Multi-Branch Sync

- Integration Service uses event log and incremental sync (change streams) to replicate branch-level transactions to central DB or reconcile during off-peak windows with conflict resolution rules.

### 4. Key Design Patterns & Guarantees

- **ACID transactions for critical ops:** Use Postgres transactions for sale+stock decrement to ensure consistency.
- **Event-driven decoupling:** Message bus for resilience—downstream failures don't block core transactions.
- **Optimistic concurrency / distributed locks:** Protect inventory updates (Redis locks or DB row versioning).
- **Idempotency keys:** For external retries (payment callbacks, supplier feeds).
- **Role-based access control (RBAC):** Enforce least privilege for all endpoints.
- **Auditability & immutability:** Every financial transaction, stock adjustment, prescription access is logged with user, timestamp, and rationale.

## 5. Security & Compliance Highlights

- TLS everywhere; HSTS for web endpoints.
- Encrypt data at-rest and in-transit; DB encryption & encrypted backups.
- Secrets via managed secret store (Vault / cloud secret manager).
- Strong authentication: JWT + rotating refresh tokens; optional MFA for admins.
- Pseudonymize or encrypt PII/PHI in DB as per local regulations; configurable data-retention policies.
- Regular vulnerability scans and dependency checks; incident response playbook.

## 6. Scalability, Availability & Reliability

- **Scaling:** Stateless backend containers scale horizontally behind load balancers. DB uses read replicas for reporting; write scale handled through efficient transaction design.
- **Availability:** Multi-AZ deployments, automated failover for DB (managed RDS).
- **Backups & DR:** Daily snapshots, point-in-time recovery, export of audit logs to immutable storage.
- **Monitoring & SLA:** Health checks, auto-healing pods, and alerting for degraded performance; define SLOs for API latency and system uptime.

## 7. Offline & Edge Considerations (for low-connectivity stores)

- Implement a lightweight offline POS mode (local IndexedDB / SQLite on device) with transaction log and robust reconciliation on reconnection.
- Conflict resolution rules: prefer server timestamp for financial ledger; alert for manual reconciliation if conflicts detected.

## 8. Integrations & Extensibility

- Third-party: SMS/Email (Twilio/SendGrid), Payment gateways, Supplier APIs, Accounting/GST exports.
- ML / Forecasting: Analytics Service can feed anonymized sales data to forecasting models (separate ML pipeline).
- API-first design: well-documented OpenAPI spec for partners and future integrations (supplier portals, e-pharmacy networks).

## **9. Recommendations & Next Steps (priority)**

1. Prototype core transaction flow: POS → Sale → Inventory decrement with full transactional safety and event emission.
2. Implement robust audit logging for all financial and prescription actions.
3. Design and test offline POS reconciliation.
4. Define role matrix and initial RBAC rules for MVP.
5. Deploy observability stack from day one.