

PHP Ledger – System Documentation

Introduction

This document provides a detailed explanation of the PHP Ledger system. The goal is to make it easy for team members and stakeholders to understand the structure, processes, and workflows of the system. The PHP Ledger is designed as a multi-company, multi-location accounting and point of sale (POS) system built on the LAMP stack (Linux, Apache, MySQL, PHP) with MeekroDB as the database abstraction layer. The system emphasizes simplicity, auditability, and scalability while ensuring offline and online operation capability.

System Architecture Overview

The system is designed as a **multi-layer architecture** to handle **multi-company, multi-branch, multi-currency** operations with strong security and offline/online synchronization.

a) Core Layers

1. Presentation Layer (Frontend)

Technologies: **Bootstrap 5 + jQuery (MVP)** (later React/Vue can be added).

Provides **separate panels** for different roles:

- **Admin Panel** → Global management
- **Store Panel** → Branch/Location management
- **Customer Panel** → Shopping, orders, payments
- **Auditor Panel** → Read-only financial reports

2. Application Layer (Backend / Business Logic)

- Technology: **Vanilla PHP + MeekroDB ORM**
- Organized into **controllers** (Auth, Company, Journal, Reports, Payments, Sync).
- Handles:
 - Authentication & RBAC (role-based access control)
 - Journal entries, COA (Chart of Accounts), reports
 - Order & product management
 - Payment processing (Meezan Bank, JazzCash, Stripe, etc.)

- Data synchronization rules

3. Middleware Layer

- Ensures **security and data isolation**:
 - **Auth Middleware** → Validates user login/token
 - **Company Scope** → Every query scoped by company_id + location_id
 - **RBAC Guard** → Enforces role permissions (admin, accountant, viewer, etc.)
 - **Open Period Guard** → Prevents journal posting into closed accounting periods

4. Data Layer (Databases)

- **Branch Local DB** (for offline mode)
- **HQ Master DB** (central ERP database)
- All accounting & transaction data stored with **UUIDs + company_id + location_id**

B) Synchronization Layer

- Sync Engine (Middleware Service) connects Local Branch DB ↔ HQ API ↔ HQ Master DB.
- Handles offline transactions, conflict resolution, incremental sync, audit logging.

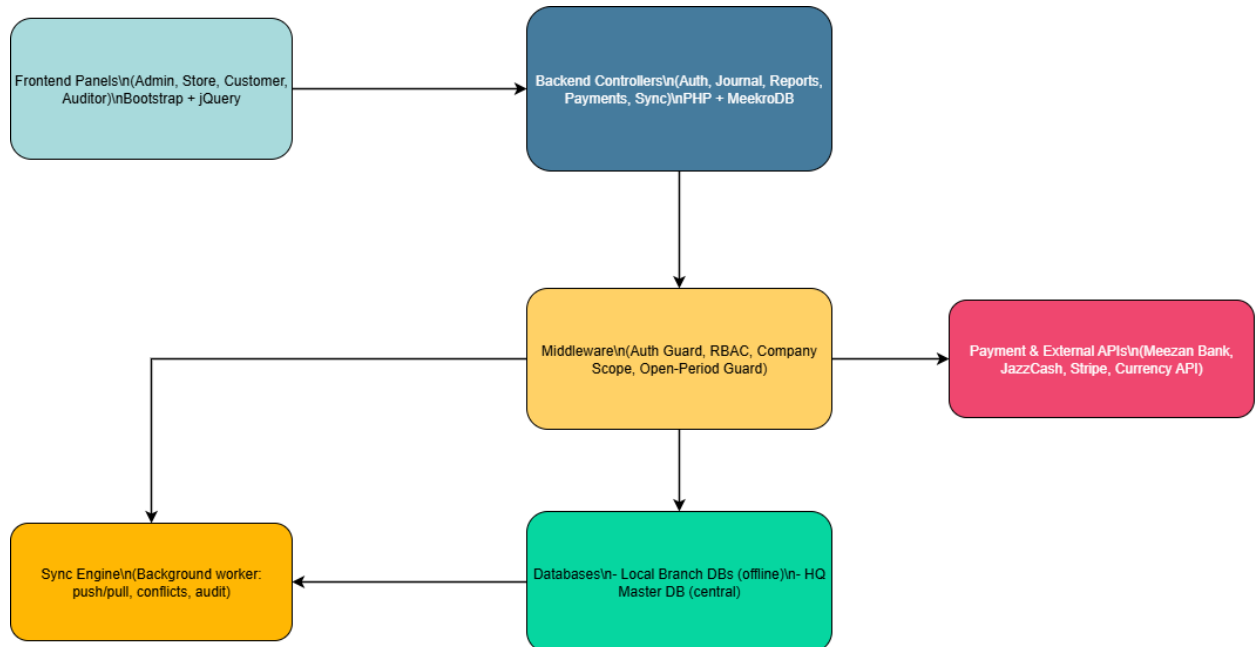
c) Security Layer

- Authentication: Password hashing (bcrypt/argon2), JWT/Session tokens, CAPTCHA + OTP (Meezan-style).
- Authorization: Role-based access (RBAC).
- Data Protection: HTTPS/SSL, CSRF protection, SQL injection prevention.
- Audit Logs: Every sensitive action recorded (login, posting, payments).

e) Example Flow

1. A **Customer in Germany** logs in → sees German products in **EUR**.
2. Places an order → local branch DB saves transaction.

3. **Sync Engine** pushes order to HQ → HQ updates central ledger.
4. **Admin** logs in HQ Panel → sees all co



Data Synchronization

1. Why Synchronization is Needed?

- Branches may work offline when internet is down.
- HQ must always have a master record of sales, inventory, and accounts.
- Different branches may update the same product/ledger at different times.
- Users in Germany, UK, Pakistan must see their own localized prices, stock, and reports.

Synchronization Architecture

1. Local Database (Branch Level)

- Each store has its own local MySQL/MariaDB.
- Transactions are stored with a UUID (unique ID) to avoid duplicate conflicts.
- Status flags: pending, synced, conflict.

2. Sync Engine (Middleware Service)

- Runs in the background.
- Periodically sends unsynced records to HQ via API.

- Pulls updates from HQ (e.g., new prices, global discounts, currency exchange rates).

3. HQ Master Database

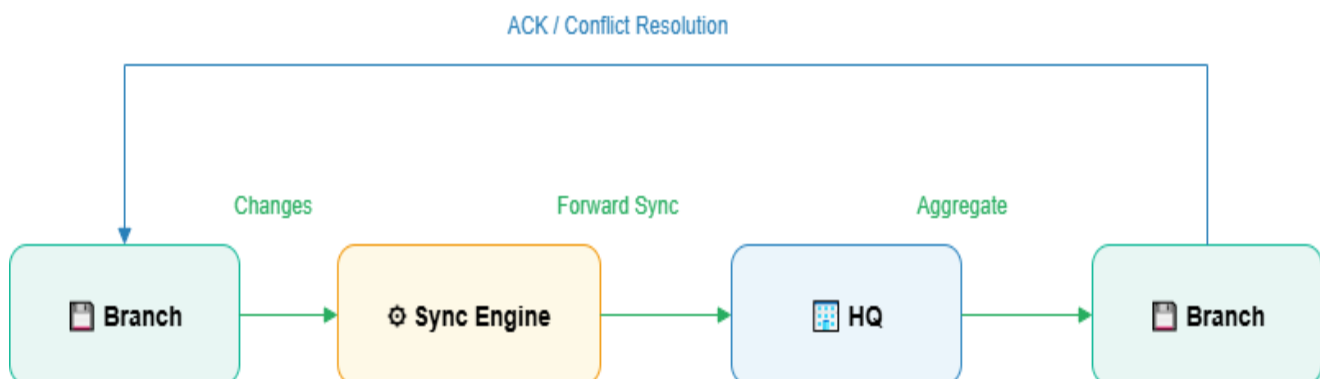
- Receives data from all branches.
- Runs validation & conflict resolution rules.
- Pushes back confirmed records to branches.

Synchronization Rules

- **Conflict Resolution**
 - Last Write Wins → Latest timestamp overwrites.
 - HQ Wins → HQ data is always final.
 - Merge Rules → For inventory, add/subtract stock changes.
- **Currency Sync**
 - HQ defines master prices in base currency (USD or PKR).
 - Branches get localized conversion (EUR for Germany, GBP for UK, PKR for Pakistan).

4. Example Flow

1. Branch A (Offline) sells 10 units of Item X.
 - a. Saved in local DB with UUID = TXN1234 and status = pending.
2. Internet restores → Sync Engine sends TXN1234 to HQ.
3. HQ API validates → updates central ledger, changes status = synced.
4. At the same time, Branch B (Online) sells 5 units of Item X.
5. HQ merges both → Final stock = (Global stock – 15).
6. HQ pushes updated stock + pricing back to both branches.



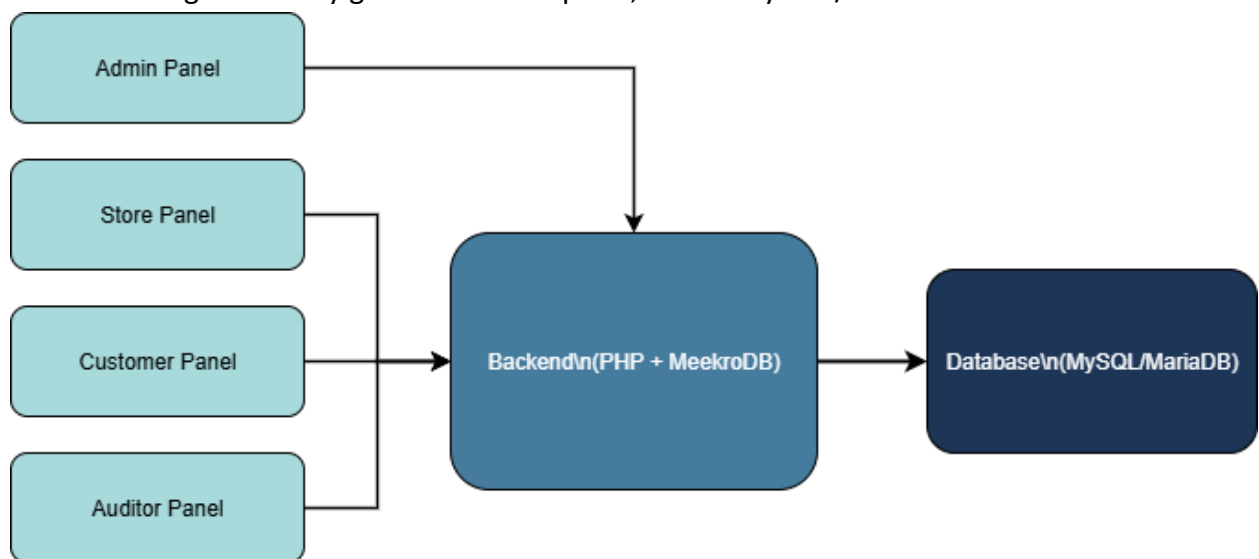
Handling Multiple Panels

How Multiple Panels Work Together

- **Authentication & Role-Based Access Control (RBAC)**
When a user logs in, the system checks their **role** (Admin / Store Owner / Customer / Auditor).
Based on that role, the user is redirected to the correct **panel**.
- **Scoped Access**
 - **Admins** → Can see and manage **everything**.
 - **Store Owners** → Data is filtered by their **company_id / location_id**.
 - **Customers** → Only see their own orders & products available in their region.
 - **Auditors** → Read-only views, export options.

c) Example Flow

- A **German customer** logs in → sees products in **EUR** with German pricing.
- A **UK store owner** logs in → sees only their **UK store orders, stock, and reports**.
- A **Pakistan Admin** logs in → sees **all global data** across companies in **PKR, GBP, EUR** as required.
- An **Auditor** logs in → only gets access to reports, not to any edit/delete functions.



Role-based access enforced in middleware (RBAC)

Authentication & Security

This is the **core layer** that ensures only **trusted users** can access the system, and that sensitive data (like transactions, payments, reports) stays safe.

Authentication Flow

- **Login Credentials**
User enters **email/username + password**.
- **CAPTCHA Verification**
 - Before processing login, system shows **reCAPTCHA / custom CAPTCHA**.
 - Prevents bots and automated brute-force attacks.
 - Example: “I’m not a robot” or image/text CAPTCHA.
- **Multi-Factor Authentication (MFA / 2FA)**
 - After CAPTCHA + password verification, send an **OTP (One-Time Password)** to mobile/email.
 - This is exactly like **Meezan Bank Mobile App** or **JazzCash app**.
 - Only after OTP is verified, system grants access.

Security Enhancements

- **Strong Password Policy** (min 8–12 chars, mix of upper/lowercase, number, special character).
- **Account Locking**: After 5 failed login attempts, account is temporarily locked.
- **CAPTCHA on Retry**: CAPTCHA becomes **mandatory** after 2–3 failed attempts.
- **Device Binding (Optional)**: Like Meezan Bank, system can remember **trusted devices** and trigger **extra verification** when login happens from a new device/location.
- **Session Timeout**: Automatic logout after inactivity.

3. Example Login Flow

1. User enters **username + password**.
2. System shows **CAPTCHA** (must solve correctly).

3. If valid → system sends **OTP (SMS/Email/App Push)**.
4. User enters OTP → system verifies.
5. If all good → generate **JWT/Session Token** and redirect to correct **panel** (Admin, Store, Customer).
6. Audit Log entry: *"User X logged in at 10:25 PM from Karachi, IP ..."*.

Localization & Multi-Currency Support

What Localization & Multi-Currency Means

- **Localization** = Making the software adapt automatically to a user's **country, language, and culture**.
- **Multi-Currency Support** = Showing **prices in the correct local currency** (EUR for Germany, GBP for UK, PKR for Pakistan, etc.).

How It Works in Your System

1. Region Detection

The system figures out where the user is from using:

- **IP-based GeoLocation** (detect country from IP address).
- **User account setting** (if the user selects their preferred region at registration).

2. Currency Display

- If user is in **Germany** → Show prices in **EUR (€)**.
- If user is in **UK** → Show prices in **GBP (£)**.
- If user is in **Pakistan** → Show prices in **PKR (Rs)**.

3. Language Localization (i18n)

- The system supports **multiple languages** (English, German, Urdu).
- Based on region or user preference, the interface changes language.
- Example:
 - German user sees **"Preis: 20 €"**
 - UK user sees **"Price: £20"**
 - Pakistani user sees **"قیمت: 2000 Rs"**

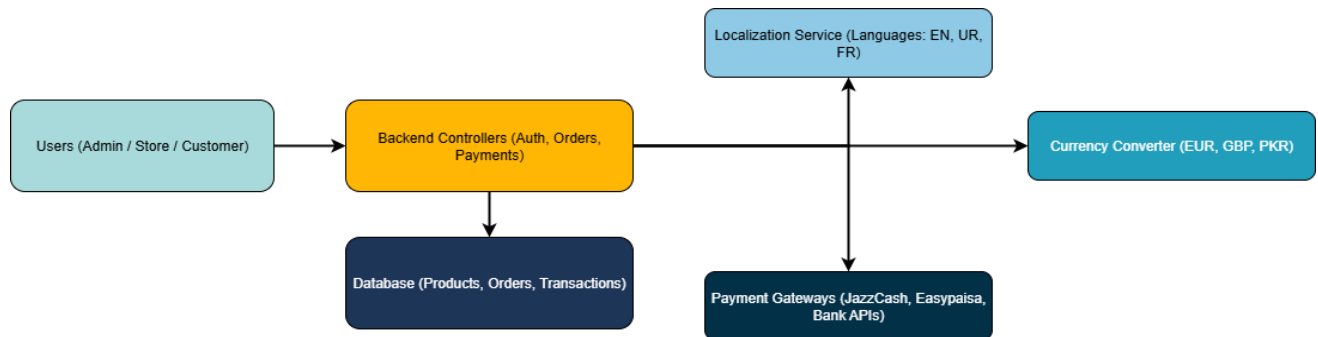
4. Currency Conversion

- The system can integrate with a **currency exchange API** (e.g., OpenExchangeRates, XE, Forex API).

- Prices can be automatically converted from a **base currency** to the **user's local currency**.

5. Admin Control

- Admin can set **default currency per store**.
- Admin can also define **fixed conversion rates** (if they don't want real-time API).



Tools & Technologies

| Component | Technology / Tool | Description |
|-----------------|-----------------------------|--------------------------------|
| Backend | PHP + MeekroDB ORM | Lightweight ORM for PHP/MySQL. |
| Database | MySQL / MariaDB | Local & central DB. |
| Offline Storage | SQLite / MySQL | Local branch storage. |
| APIs | REST (JSON) | Branch-HQ communication. |
| Queue | RabbitMQ / Redis (optional) | Async sync handling. |
| Security | JWT Auth + SSL | Secure sync & APIs. |
| Frontend | React.js / Bootstrap | Dashboard & analytics. |

Conclusion

This software architecture is designed to be **modular, scalable, and secure**, addressing the key requirements of multi-database management, multi-panel interfaces, localization, and real-time synchronization. By separating concerns across the

presentation, application, and data layers, the system ensures flexibility for future expansion, such as integrating new regions, currencies, or business modules (e.g., AR/AP, POS, or ERP extensions).

Key takeaways:

- **Multi-database + regional replication** → guarantees data isolation, availability, and resilience.
- **Role-based panels (Admin, Store, Customer)** → provide clear separation of responsibilities and secure access.
- **Localization & currency handling** → ensure users always see relevant pricing, language, and regional content.
- **Strong authentication & security controls** → increase customer trust and safeguard sensitive financial data.