SYSTEM ARCHITECTURE & TECHNICAL DOCUMENTATION

Laikipia County ICT Equipments Ledger System (v2.4)

System Type:	Inventory Management & Asset Lifecycle Tracking Platform
Core Tech Stack:	PHP 8.x, PDO (PHP Data Objects), MySQL InnoDB, Tailwind CSS, JavaScript Engine
Security Protocol:	BCRYPT Standard Hashing, Session Hijacking Prevention, Dual-Layer RBAC Firewalls
Target Environment:	County Government of Laikipia — ICT Department Network infrastructure

1. Executive & Functional Overview

The Laikipia County ICT Equipments Ledger System is a secure, lightweight enterprise asset-tracking application engineered to manage, monitor, and audit the lifecycle of county computing hardware. The application shifts asset logging away from generic spreadsheets to a stateful database, mitigating asset loss, optimizing resource distribution tracking, and establishing non-repudiation bounds.

The system relies on a structural dual-transaction process protocol: **Receiving (IN)** and **Issuing (OUT)**. All actions are cataloged against specific users, minimizing inventory tracking degradation.

2. System Architecture & Component Mapping

The platform utilizes a structured architecture where client-side interactions submit states to server-side engines that execute transaction-wrapped operations against an atomic database matrix.

2.1 core Architecture Files

- **`config.php`**: Establishes global parameters and boots the primary PDO database connection handle.
- **`login.php`**: Performs credential extraction, BCRYPT validation, session identifier regeneration, and loads user operation metadata into the global session.
- **`index.php`**: The client-facing input panel. Responsively toggles layout states based on user privilege metrics.

- **`process.php`**: The core database transaction processing engine. Intercepts inputs, checks rules, executes database logs, and handles failures cleanly.
- **`admin.php / dashboard.php`**: Management dashboard providing analytics, ledger search tools, and user access profile management.
- **`logout.php`**: Completely clears active server footprints and returns users to the login screen.

3. Database Schema Design

The system runs on two interconnected foundational database entities using the MySQL InnoDB relational storage engine to ensure transactions remain ACID (Atomicity, Consistency, Isolation, Durability) compliant.

3.1 The `system_users` Entity Schema

Stores personnel access parameters, profile records, and permission limitations.

Column Name	Data Type	Constraints / Indices	Functional Context
`id`	INT	PRIMARY KEY, AUTO_INCREMENT	Internal unique user indicator.
`username`	VARCHAR(50)	UNIQUE INDEX	Sanitized security login identifier string.
`password`	VARCHAR(255)	NOT NULL	Cryptographically secure BCRYPT standard hash value.
`fullname`	VARCHAR(100)	NOT NULL	Officer's true identification string.
`role`	VARCHAR(20)	DEFAULT 'storekeeper'	Access level tiers: `'admin'` or `'storekeeper'`.
`department`	VARCHAR(100)	NOT NULL	Personnel administrative workspace mapping.
`designation`	VARCHAR(100)	NOT NULL	Human Resource structural job role signature.
`station`	VARCHAR(100)	NULLABLE	Physical regional deployments location.
`allowed_action`	VARCHAR(20)	DEFAULT 'BOTH'	RBAC token bounds: `'BOTH'`, `'RECEIVE'`, `'ISSUE'`.

3.2 The `store_ledger_v2` Transaction Entity Schema

Maintains absolute historical tracking parameters for each incoming or outgoing piece of equipment.

Column Name	Data Type	Operational Context
`id`	INT (PK)	Primary tracking transactional index reference key.
`transaction_type`	VARCHAR(20)	Operational workflow state code: `'RECEIVING'` or `'ISSUING'`.
`entry_date`	DATE	The physical or formal date the ledger event took place.
`product_name`	VARCHAR(150)	Manufacturer item specification and identifier model nomenclature string.
`serial_number`	VARCHAR(100)	Unique structural hardware string identifier tag. Forced UPPERCASE.
`is_available`	TINYINT(1)	Boolean tracking metric for physical presence inside storage facility boundaries.

4. Multi-Tier Security Matrix (RBAC Firewall Architecture)

Security is implemented across three distinct layers to defend against execution bypasses and cross-boundary privilege escalation.

4.1 Layer 1: Authenticated Session Routing

Every entry file utilizes security validation guards that intercept incoming global server requests before executing any underlying functionality:

```
if (!isset($_SESSION['user_id'])) {
    header("Location: login.php");
    exit();
}
```

4.2 Layer 2: Dynamic UI Privilege Obfuscation

The interface on the `index.php` control dashboard monitors the context variables loaded during initial authentication. If an officer's profile is restricted to `RECEIVE` or `ISSUE`, the alternating button element layout option is dynamically omitted from the DOM via PHP.

4.3 Layer 3: Server-Side Processing Firewall

Because frontend code can be manipulated in the browser via Developer Tools (F12), `process.php` executes a mandatory back-end check. This check intercepts requests and aborts execution if a user tries to submit an operation that violates their assigned database privileges:

```
$allowedAction = $_SESSION['allowed_action'] ?? 'BOTH';
$type = $_POST['transaction_type'] ?? 'RECEIVING';

if ($allowedAction === 'RECEIVE' && $type === 'ISSUING') {
    header("Location: index.php?status=error&msg=" . urlencode("Access Denied: Restricted to RECEIVING tasks."));
    exit();
}
```

5. Transaction Execution & Core Logic

To preserve data integrity, the ledger system implements strict asset lifecycle verification rules within isolated database operations.

5.1 The Ledger Availability Calculus

The availability status of any hardware item is tracking-dependent and calculated by tracking changes over time. Let A denote item availability inside store parameters, where:

$$A \in \{0, 1\}$$

When processing a new hardware entry, the transaction framework applies specific business rules based on the operation type:

1. **`RECEIVING` Flow**: Checks if the serial number is already registered with an active status ($A = 1$). If found, the transaction is rejected to prevent duplicate logging errors.
2. **`ISSUING` Flow**: Validates that the requested hardware item is currently in stock ($A = 1$). If it is not, the operation is blocked to prevent issuing the same asset multiple times.

5.2 Isolated Database Transactions

When an item is issued, the application opens a database transaction block, logs the transaction event, and immediately updates the availability status of all previous matching records to 0:

```
$pdo->beginTransaction();
// [Operation A: Write fresh record entry row into history log table]
// [Operation B: Update inventory status]
$updateSql = "UPDATE store_ledger_v2 SET is_available = 0 WHERE serial_number = ?";
$pdo->commit();
```

6. Administrator Control Panel & User Management

The `admin.php` dashboard serves as the administrative control center. It handles tracking analytics, printable inventory audits, and managing system officer profiles.

6.1 Dynamic User Management Framework

The administration panel includes a dynamic, dual-purpose profile management console. Through a JavaScript interaction layer, the workspace form switches between a registration interface and a profile editor without requiring page reloads.

- **Profile Initialization**: By default, the panel functions as an account registration interface, requiring a full password setup to create new user records.
- **Form Transformation for Editing**: Clicking `Edit` calls the `populateOfficerEditForm()` function, which populates the form field values with the officer's current profile data using data attributes.
- **Password Preservation Logic**: When updating a user profile, the password field becomes optional. If left blank, the system updates the profile records while keeping the existing password hash intact.

6.2 Technical User Management Controls

Action Code Trigger	Processing Logic & Security Validations	System Outcome Status
`register_officer`	Validates unique username requirements and hashes the password string using the standard BCRYPT protocol.	Inserts a new user record into the `system_users` directory database table.
`update_officer`	Extracts the active record target ID via POST parameters. If a new password string is provided, it updates the password hash; otherwise, it preserves the current hash value.	Updates the user's profile information, department alignment, and module privileges.
`action_delete`	Verifies session authenticity parameters and executes a strict record deletion command using prepared statements.	Permanently drops the specified target transaction entry row from the `store_ledger_v2` table.

7. Troubleshooting, Diagnostics & Maintenance

When performing system updates or debugging database operations, refer to this diagnostic checklist:

1. **Permission Sync Errors**: If an officer's access level changes in the database but their interface layout doesn't update, have the user log out and log back in. Access levels are loaded into the session during the authentication process.
2. **PDO Connection Faults**: If the system throws an unhandled error indicating that the `$pdo` variable cannot be found, verify that `config.php` is properly configured and successfully included within the active execution context.
3. **Database Integrity Violations**: If transaction operations return error code `23000`, check for a duplicate entry conflict, such as trying to register a username that already exists in the system.