

Document & Inventory Monitoring System (DMS IMS): Relational Database-Driven Intranet System for High-Integrity Asset Tracking in Educational Institutions

William C. Dionisio, Alison Ephraim Medina, Brianne Singson, Ma. Magdalena V. Gatdula, DT.

Graduate School, Bulacan State University, City of Malolos, Bulacan, Philippines

DOI: <https://doi.org/10.51583/IJLTEMAS.2026.150900112>

Received: 27 September 2026; Accepted: 02 October 2026; Published: 19 October 2026

ABSTRACT

Managing institutional stockrooms through manual spreadsheets routinely produces phantom inventory, untraceable edits, multi-user update collisions, and a complete loss of accountability. To address these failures, this study designed, developed, integrated, and evaluated the Document & Inventory Monitoring System (DMS IMS), a secure, high-integrity web application built for local intranet deployment. Developed through an Agile Scrum process across seven two-week sprints, the system pairs a Laravel 9.x model-view-controller (MVC) backend with a MariaDB schema normalized to Third Normal Form (3NF) and a Bootstrap 5.3 frontend using asynchronous JavaScript and XML (AJAX). Core features include automated stock-keeping unit (SKU) generation, bcrypt credential hashing (work factor 12), an append-only audit trail, and ACID-compliant (Atomicity, Consistency, Isolation, Durability) Stock-In/Stock-Out transactions. Evaluation followed a two-stage protocol: developer-led unit and integration testing, then a ten-case system suite (ST-001 to ST-010) on an isolated local area network (LAN) simulating up to 50 concurrent connections. Initial runs achieved a 70% pass rate; three defects (a concurrency lock failure, a search-latency breach, and a valuation rounding error) were resolved through column indexing, transactional retries, and strict decimal types, yielding a 100% final pass rate. Under peak load, live-search latency fell from 1.9 seconds to 0.4 seconds, exceeding the 1.2-second requirement. DMS IMS offers public institutions a low-cost, locally hosted asset-tracking platform that keeps institutional data under institutional control without internet dependence. The findings support further pilot deployment and usability validation with actual staff.

Keywords: inventory monitoring system, Agile Scrum Software Development Life Cycle (SDLC), relational database integrity, software quality testing, software engineering.

INTRODUCTION

Accurate resource tracking is essential for public universities and enterprise organizations. Yet many institutions still rely on manual logbooks or shared electronic spreadsheets to manage physical assets. In academic and government settings, software engineering demands verifiable requirements, structured development, and evidence-based testing. Under ISO/IEC/IEEE 29148 standards (International Organization for Standardization, International Electrotechnical Commission, & Institute of Electrical and Electronics Engineers [ISO/IEC/IEEE], 2018), software requirements must remain unambiguous and traceable across every phase of development, while ISO/IEC 25010 (International Organization for Standardization & International Electrotechnical Commission [ISO/IEC], 2011) mandates measurable quality standards in functionality, performance, reliability, and security. In this project, our team took an initial software concept and turned it into a fully working system over four structured milestones.

At Bulacan State University (BulSU) and similar public schools, staff frequently rely on Microsoft Excel files for inventory records. This approach creates severe operational bottlenecks: spreadsheets lack transaction isolation, allow inconsistent category-location entries, leave no edit history, and invite data corruption when two people edit the file simultaneously. When multiple staff members update stock counts at once, 'phantom inventory' and double-deductions occur, corrupting stock totals. System administrators and departmental staff

need accurate real-time stock balances, tamper-proof activity logs, and reliable reporting. Commercial cloud systems exist, but heavy subscription fees and constant internet requirements make them impractical for many local campuses.

A locally hosted intranet application fits public school settings for three practical reasons. First, campus internet connectivity can be spotty or slow, causing cloud tools to fail during high-volume periods like enrollment or annual audits. Second, recurring cloud licensing fees strain annual government procurement budgets, whereas a self-hosted platform runs on existing local network hardware at no additional software cost. Third, Philippine public institutions must comply with the Data Privacy Act of 2012 (Republic Act No. 10173, 2012); storing asset data on institutionally controlled local servers keeps records secure and eliminates third-party data custody risks. To address these needs, we engineered the Document & Inventory Monitoring System (DMS IMS), a web-based asset tracking tool running on a normalized relational database and designed specifically for local intranet deployment.

The primary objective of our study was to design, develop, integrate, and test the DMS IMS prototype to replace error-prone spreadsheet tracking. Specifically, we set out to:

1. Implement a secure login gateway using bcrypt password hashing (work factor 12) and role-based access control (RBAC) middleware.
2. Structure normalized, zero-redundancy master tables for categories and storage locations following Third Normal Form (3NF) rules.
3. Program an automated stock-keeping unit (SKU) code generator (e.g., OFFS-LOC-0001) with unique database indexing and administrator manual overrides.
4. Construct a Stock Movement Ledger using atomic, 'all-or-nothing' database transactions that validate stock levels before processing dispatches.
5. Build an interactive dashboard displaying live total portfolio values in Philippine Pesos (₱) alongside visual low-stock warnings.
6. Create an asynchronous reporting engine that generates print-ready PDF summaries stamped in Philippine Standard Time (PST) and exports raw data to Excel.
7. Implement an append-only, chronologically indexed audit log capturing every database change with active user IDs and client IP addresses.
8. Evaluate system performance, route security, and data integrity using a 10-case system test suite across an isolated Local Area Network (LAN).

The scope of the DMS IMS prototype is designed for single-site intranet deployment over a LAN. It covers user roles, master category and bay location setup, item catalog management, real-time stock transactions, visual dashboard indicators (key performance indicators, KPIs), PDF/Excel report generation, and security audit logging. Several specific boundaries apply:

- Deployment is limited to local web server environments (such as XAMPP or Laragon) accessed via modern desktop browsers (Chrome, Edge, Firefox), without native mobile apps.
- Financial accounting functions, such as straight-line or declining-balance depreciation are not included.
- Physical item condition tracking (active, damaged, condemned) is outside the current scope.
- Asset valuations reflect original procurement purchase prices without separate appraisal adjustments.
- Real-time database replication across multiple physical campuses is not supported.

REVIEW OF RELATED LITERATURES AND STUDIES

To build a strong foundation for DMS IMS, we reviewed forty-one (41) scholarly and technical sources organized into five thematic areas: (1) manual recording risks and spreadsheet error mitigation; (2) Agile Scrum development frameworks; (3) secure web application design and access control; (4) relational database integrity, normalization, and atomic transactions; and (5) software testing, latency benchmarks, and ISO quality models.

First, the foundational software crisis discussed at the 1968 NATO conferences (Naur & Randell, 1969) highlighted the essential difference between casual coding and disciplined software engineering (Sommerville, 2016). In inventory management, paper logbooks and disconnected spreadsheets create severe operational risks. Tompkins et al. (2010) demonstrated that manual data entry leads to frequent transcription errors, lacks concurrent access controls, and allows untraceable edits. Panko (1998) found that almost all operational spreadsheets contain errors due to a lack of database-level validation, a weakness that directly impacts inventory ledgers stored as flat files. To solve these vulnerabilities, Pressman and Maxim (2020) recommended centralized web applications that enforce strict input validation at the database level, ensuring full data accountability and institutional control (Nielsen, 1994).

Second, modern software development emphasizes iterative delivery. The Agile Manifesto (Beck et al., 2001) shifted industry practice away from rigid linear approaches like the Waterfall model (Royce, 1970), prioritizing working software, stakeholder feedback, and flexible sprint cycles instead. Scrum frameworks (Schwaber & Sutherland, 2020) break development into time-boxed sprints, helping teams catch requirement mismatches early. Cohn (2004) showed how user stories effectively map operational needs into prioritized product backlogs, while Boehm (1988) demonstrated that iterative prototyping manages risk in high-stakes projects. Research by Forsgren et al. (2018) confirmed that automated testing and continuous integration accelerate delivery and improve software stability, building on core continuous delivery principles (Humble & Farley, 2010).

Third, system security depends heavily on application architecture. According to the OWASP Foundation (2021a), broken access control and SQL injection remain leading web security threats. Protecting state-changing web forms requires cryptographic Cross-Site Request Forgery (CSRF) tokens (OWASP Foundation, 2021b). For credential protection, the National Institute of Standards and Technology (NIST, 2022) recommends adaptive hashing algorithms like bcrypt or Argon2 to block offline brute-force attacks, guidelines rooted in fundamental cryptographic principles (Diffie & Hellman, 1976). Shneiderman et al. (2016) emphasized that user interfaces should enforce role-based access by hiding restricted menu options entirely, upholding the Principle of Least Privilege in line with human-centered design standards (International Organization for Standardization [ISO], 2019). Furthermore, NIST guidelines (NIST, 2020) state that security controls must be built directly into database schemas and routing middleware rather than added as an afterthought.

Fourth, relational database design provides the structural backbone for data consistency. Codd (1970) proved that organizing data into Third Normal Form (3NF) eliminates update anomalies, redundant text entries, and transitive dependencies (Elmasri & Navathe, 2015). Date (2004) showed how referential integrity rules (such as RESTRICT and CASCADE) prevent orphaned records and maintain database structure. Gray (1981) defined the ACID properties (Atomicity, Consistency, Isolation, Durability) of database transactions, proving that wrapping multi-table operations in transaction blocks guarantees complete rollbacks if a step fails. Modern relational engines like MySQL and MariaDB (InnoDB) deliver fast read-write performance and reliable transaction handling for local servers (Kleppmann, 2017), while proper database indexing on primary and foreign keys prevents server lag during heavy search queries (Silberschatz et al., 2020).

Fifth, systematic testing verifies that software meets its technical goals. Myers et al. (2011) defined testing as a disciplined search for defects using boundary value analysis to catch edge-case errors. Bass et al. (2021) highlighted software testability, the ease with which a system reveals its faults during execution. To ensure requirements compliance, the International Software Testing Qualifications Board (ISTQB, 2023) requires clear traceability across component, integration, and system test levels. The ISO/IEC 25010 model (ISO/IEC, 2011) defines clear metrics for performance, security, usability, and maintainability. Brooke (1996) noted that usability depends entirely on the operational context of the user, while Martin (2008) showed that clean, standardized code structures keep systems modular and easy to maintain over time. Finally, the Philippine Qualifications Framework (Level 7) expects graduate engineering work to demonstrate rigorous self-directed research and professional technical execution (BulSU Graduate School, 2026).

Critical synthesis of these streams reveals three weaknesses in the prior literature that motivated this study. First, although normalization and ACID transactions are textbook practice, many published inventory-system evaluations remain feature checklists: they demonstrate that modules exist but rarely stress those modules under controlled multi-user contention, which is precisely where spreadsheet-based workflows fail. Second, standards-driven evaluations inspired by ISO/IEC 25010 typically assess a single user on a single machine, leaving concurrency behavior and peak-load latency underreported. Third, an implicit cloud-first bias runs through the literature: hosted deployments are treated as the default, so data sovereignty, recurring subscription cost, and intermittent-connectivity operation receive little direct treatment, even though these factors decide adoption in public-sector settings. There is also a practical tension worth noting: Agile methods favor flexibility while public procurement favors documentation and traceability; this study reconciles the two by pairing a Scrum workflow with a formal Software Requirements Specification (SRS) and an explicit requirement-to-test traceability matrix.

While database normalization and ACID transactions are well-established principles, most commercial asset tracking platforms are built as cloud-first applications requiring continuous internet connections and subscription fees. Few solutions offer a lightweight, intranet-focused application designed for local school networks. Table 1 compares DMS IMS with two widely used open-source inventory systems, Snipe-IT (Snipe-IT, n.d.) and PartKeepr (PartKeepr, n.d.). Both are mature self-hosted alternatives, but both were designed for IT asset or electronic parts management and default to configurations that assume broader network availability; neither provides built-in Philippine Peso valuation, Philippine Standard Time reporting, or an append-only, user-attributed audit trail aimed at local institutional audits. As Davis (1989) noted in the Technology Acceptance Model (TAM), user adoption relies heavily on perceived usefulness and ease of use; DMS IMS therefore complements its technical safeguards with plain-language interface labels, while leaving formal usability validation to future work (see Limitations). DMS IMS fills this gap by deploying a secure, locally hosted Laravel 9.x application that converts low-level database rules into simple 'all-or-nothing' transaction safety guards and 'recycle bin' soft deletes, delivering high data integrity without cloud dependence.

Table 1: Comparison of DMS IMS with Selected Open-Source Inventory Systems (features as of 2026, based on public documentation)

Criterion	Snipe-IT	PartKeepr	DMS IMS (this study)
Primary target assets	IT assets (laptops, licenses)	Electronic parts/components	General institutional inventory and documents
Deployment model	Self-hosted or cloud-hosted	Self-hosted	Self-hosted intranet only; no internet required
Localization	Configurable multi-currency	Not localized	Philippine Peso (₱) valuation and PST timestamps by default
Audit trail	Configurable change history	Basic logging	Append-only, user-attributed, IP-logged audit trail
Concurrency safeguards	Standard database transactions	Not emphasized	ACID ledger with row-level locking and automatic retry
Licensing and cost	Free, open source (MIT)	Free, open source (GPL)	Free, in-house development

Definition of Terms

- **Functional Requirement:** A specific capability, feature, or task the software must perform when requested by a user (e.g., generating a unique SKU code).
- **Non-Functional Requirement:** A measurable performance standard or technical constraint governing how well the system operates (e.g., search response speed).
- **Minimum Viable Product (MVP):** An initial, functional version of software containing only the core features needed for early user testing and feedback.

- **Module Testing:** Testing an individual feature or component in isolation before connecting it to the broader system.
- **Integration Testing:** Verifying that data flows smoothly and correctly between connected software modules.
- **System Testing:** Evaluating the complete, fully assembled software prototype against all approved functional and non-functional requirements.
- **Requirement Traceability:** The practice of linking every approved project requirement from initial design through development to direct test results.

METHODOLOGY

This study followed a developmental software engineering design guided by an Agile Scrum life cycle, aligned with ISO/IEC/IEEE 12207:2017 standards (ISO/IEC/IEEE, 2017). We chose Agile Scrum because building a multi-module database platform requires ongoing adjustments. Linear frameworks like Waterfall assume requirements remain static, which frequently leads to clashes when software meets real campus workflows. Agile Scrum resolves this by breaking work into short, two-week sprints where functional increments are built, tested, and reviewed immediately. We maintained source code using Git and GitHub and implemented a two-tier quality control loop to verify features individually before full system integration. Clean design patterns and refactoring practices were applied throughout development to keep the codebase clean and maintainable (Gamma et al., 1994; Fowler, 2018).

The project progressed across four course milestones: Activity 1 established the project charter, problem statements, and initial tooling; Activity 2 defined user roles, formulated the Software Requirements Specification (SRS), and mapped UI wireframes; Activity 3 turned static designs into working PHP modules, connected the database, and conducted module testing; Activity 4 unified all modules into a single prototype, ran ten system-level test cases, resolved identified bugs, and verified final compliance. Table 2 outlines this chronological roadmap.

Table 2: Project Timeline and Agile Sprint Roadmap

Sprint	Focus Module	Primary Developer	Target Deliverable	Operational Value (Plain-Language)
Sprints 1-2	Authentication & Master Data	Alison Medina / Brianne Singson	Secure Login; 3NF Categories & Locations	Advanced Password Security Encryption; Zero-Redundancy Storage
Sprints 3-4	Catalogue & Stock Ledger	Brianne Singson / Alison Medina	Auto-SKU generator; Ledger; AJAX search	Auto-generated Item Codes; All-or-Nothing Transactions
Sprint 5	Dashboard & Visual Alerts	Brianne Singson	Dynamic Dashboard; Real-time Peso totals	Instant Valuation (₱) and visual low-stock indicators
Sprint 6	Reporting & Multi-format Exports	William Dionisio	DomPDF script; Excel templates; CSS print	Print-ready PDF reports in PST time; Excel downloads
Sprint 7	Audit Trail & Security	William Dionisio / Alison Medina	Immutable audit logs; Route protections	Unalterable activity logs; Route access blocking
Activity 3	MVP Validation	Entire Team	5 functional module test cases (MT-001 to MT-005)	Verification of isolated ledger features and search
Activity 4	Unified Prototype & System Audits	Entire Team	10 system-level test cases (ST-001 to ST-010)	Full verification of security boundaries and transaction locks

Table 3: Requirements Specification and Priority Matrix

Req. ID	Approved Requirement Statement	Priority	Layman Validation Value
FR-1.1	The system shall authenticate users via email and encrypted password hashes.	Must Have	Advanced Login Encryption: secures accounts and scrambles password text.
FR-1.2	The system shall block unauthorized operators from administrative setup menus.	Must Have	Role-Based Access: Staff clerks cannot modify accounts or system setups.
FR-2.1	The system shall maintain isolated, 3NF normalized tables for Categories and Locations.	Must Have	Zero-Redundancy setups prevent spelling mistakes and orphaned data.
FR-3.1	The system shall automatically generate unique alphanumeric SKUs (e.g., CAT-LOC-0001).	Must Have	Auto-SKU generator prevents tracking errors and duplicates on item registration.
FR-3.2	The system shall store unit costs in DECIMAL(10,2) and calculate total value dynamically.	Must Have	Bank-grade math calculations prevent currency rounding bugs in Peso (₱).
FR-3.3	The system shall flag items at or below reorder levels with high-contrast visual alerts.	Must Have	Visual Low-Stock alerts give managers instant warning when stock needs ordering.
FR-3.4	The system shall implement soft-deletions using deleted_at timestamps.	Must Have	Recycle-Bin deletion: hides active items but preserves old historical logs.
FR-4.1	The system shall wrap ledger Stock-In/Out in atomic database transactions.	Must Have	All-or-Nothing writes: rolls back half-saved records on computer crash.
FR-4.2	The system shall validate Stock-Out requests and reject over-allocations.	Must Have	Stock check safeguards: blocks requests that exceed stock on hand.
FR-5.1	The system shall compile formal PDF reports with PST timestamps and Excel exports.	Should Have	Print-optimized official PDF and spreadsheet downloads in Philippine time.
FR-6.1	The system shall capture all state mutations in an unalterable append-only audit log.	Should Have	Unmodifiable chronological history ledger showing user ID and network IP.
FR-7.2	The system shall provide custom text inputs in ledger forms for QR stub validation.	Could Have	QR Hardware placeholder: supports scanning integration in future releases.
NFR-1.1	Page response latency for database searches and dashboards shall be ≤ 1.2 seconds.	Must Have	Performance speed: ensures rapid, lag-free operations under high traffic.
NFR-2.1	All state-changing requests shall validate secure tokens (CSRF) and parameters.	Must Have	Security Shields: blocks automated hacker scripts and form manipulations.

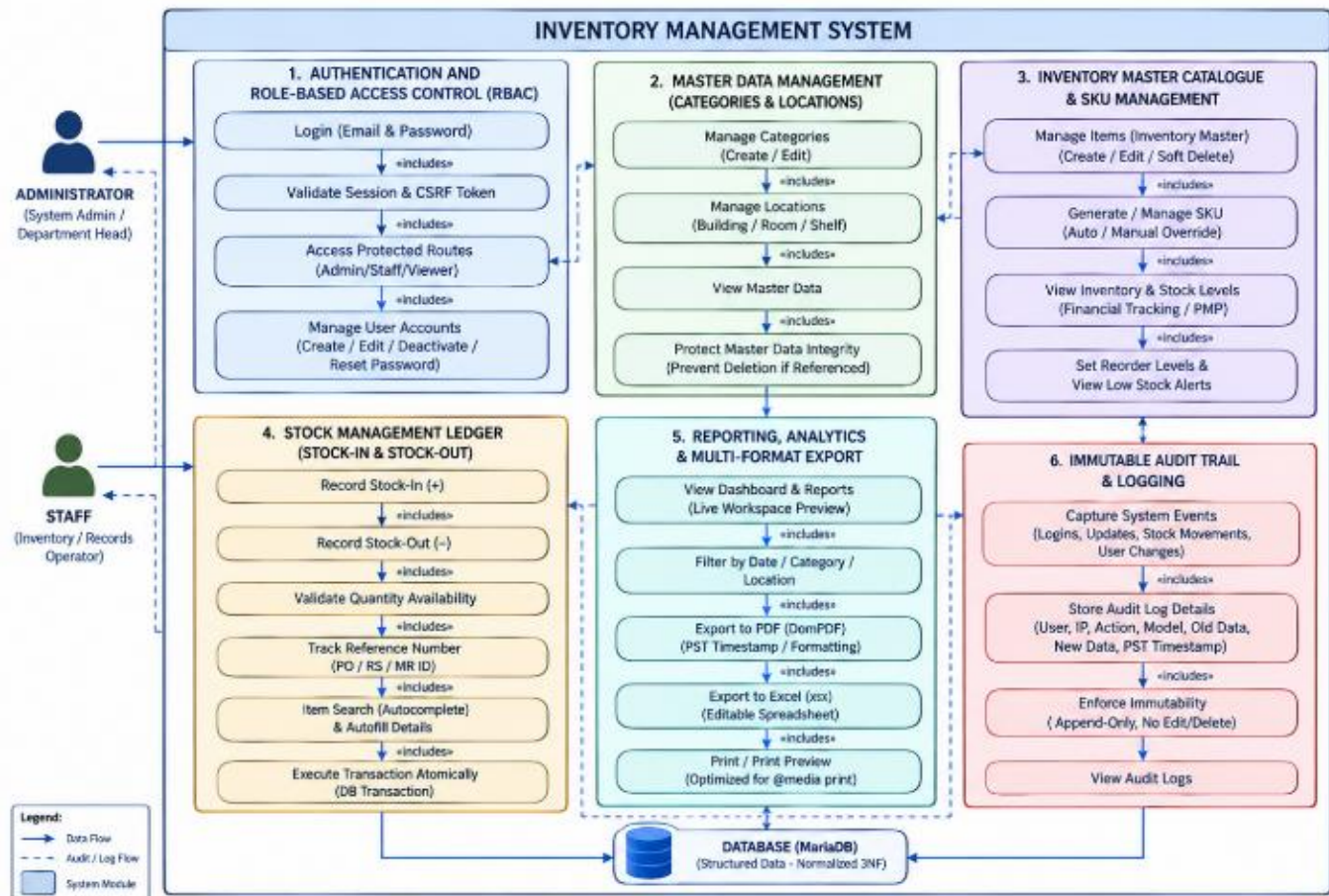


Figure 1: Use-Case Diagram

System Architecture and LAN Testing Setup

The system uses a classic three-tier client-server architecture deployed on a local intranet web server. The presentation tier uses HTML5, CSS3, and JavaScript styled with Bootstrap 5.3 in a dark 'Slate' theme using the Inter font family. The application logic tier runs on PHP Laravel 9.x following the model-view-controller (MVC) pattern, with custom middleware for role-based route access control. The database tier uses MariaDB 10.4 with full referential foreign key enforcement and 3NF schema normalization.

Instead of relying on subjective user survey forms, we evaluated system quality through a rigorous two-tier technical testing protocol. Developer-led Gate 1 testing verified individual modules during development. Quality assurance (QA)-led Gate 2 testing evaluated the integrated system across ten end-to-end test scenarios (ST-001 to ST-010), pushing the platform against unauthorized route access, invalid data entries, inventory over-allocations, simulated server crashes, and multi-user update collisions. Defects were logged, classified by severity (Critical, Major, Minor), fixed, and retested through regression passes. A sample scripted test case is provided in Appendix A.

All benchmarks were conducted on an isolated LAN testbed. The server ran Windows 10 Pro on an Intel Core i5 processor with 16 GB DDR4 RAM and SSD storage, running Laragon 6.0 with Apache 2.4, PHP 8.0, and MariaDB 10.4. Five client workstations connected to the server through a gigabit Ethernet switch. To simulate realistic campus demands, we seeded the database with 1,000 inventory items across multiple categories and 10,000 transaction log rows. Peak user load was simulated using Apache JMeter 5.6: 50 virtual users issued continuous live-search requests over a 10-minute test window. Response times were recorded at the browser network layer across three independent trials, with reported figures representing trial averages. Benchmarking was performed at this 50-user peak load only; per-load reporting at lighter loads (10 and 25 users) is identified as future work in the Recommendations.

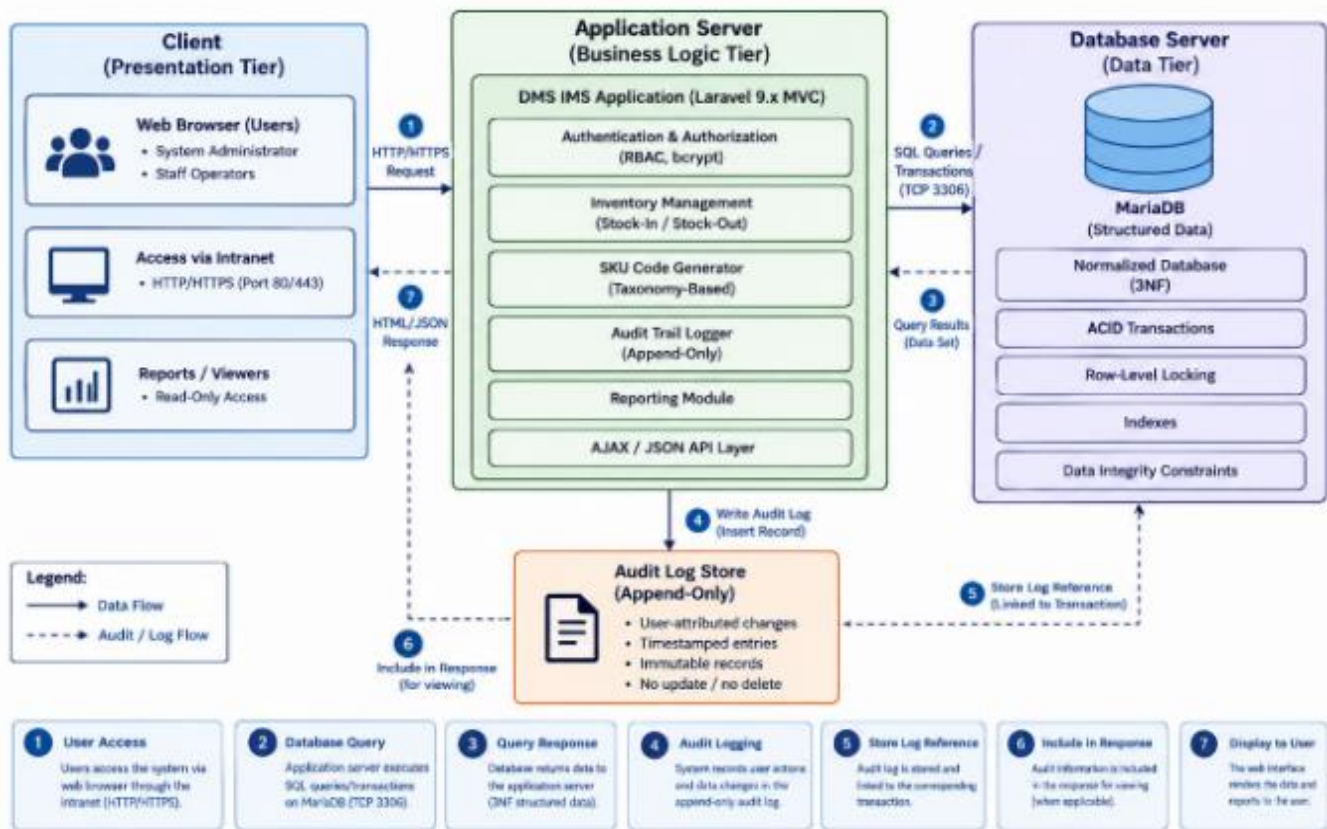


Figure 2: System Architecture

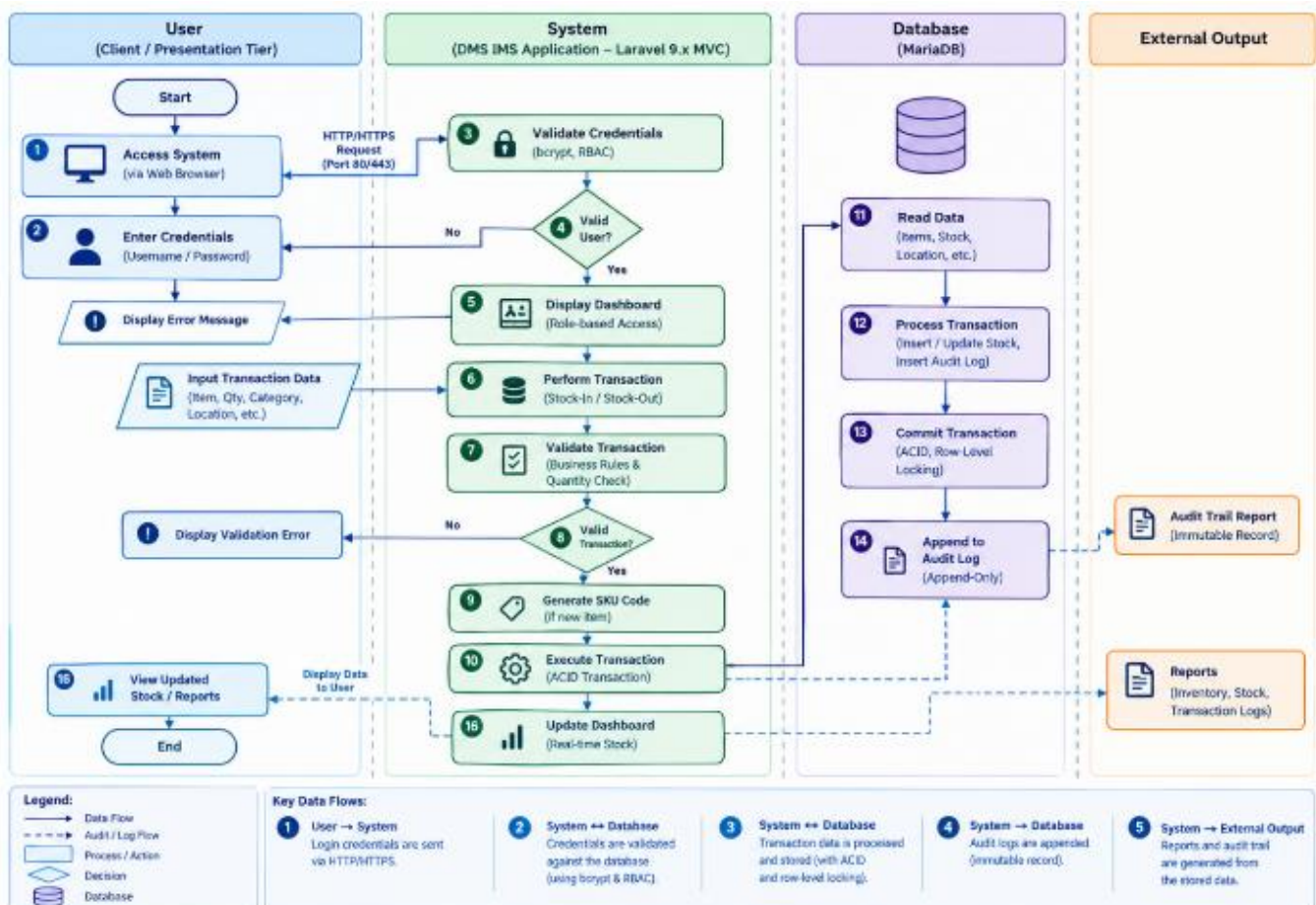


Figure 3: Detailed Process Flow

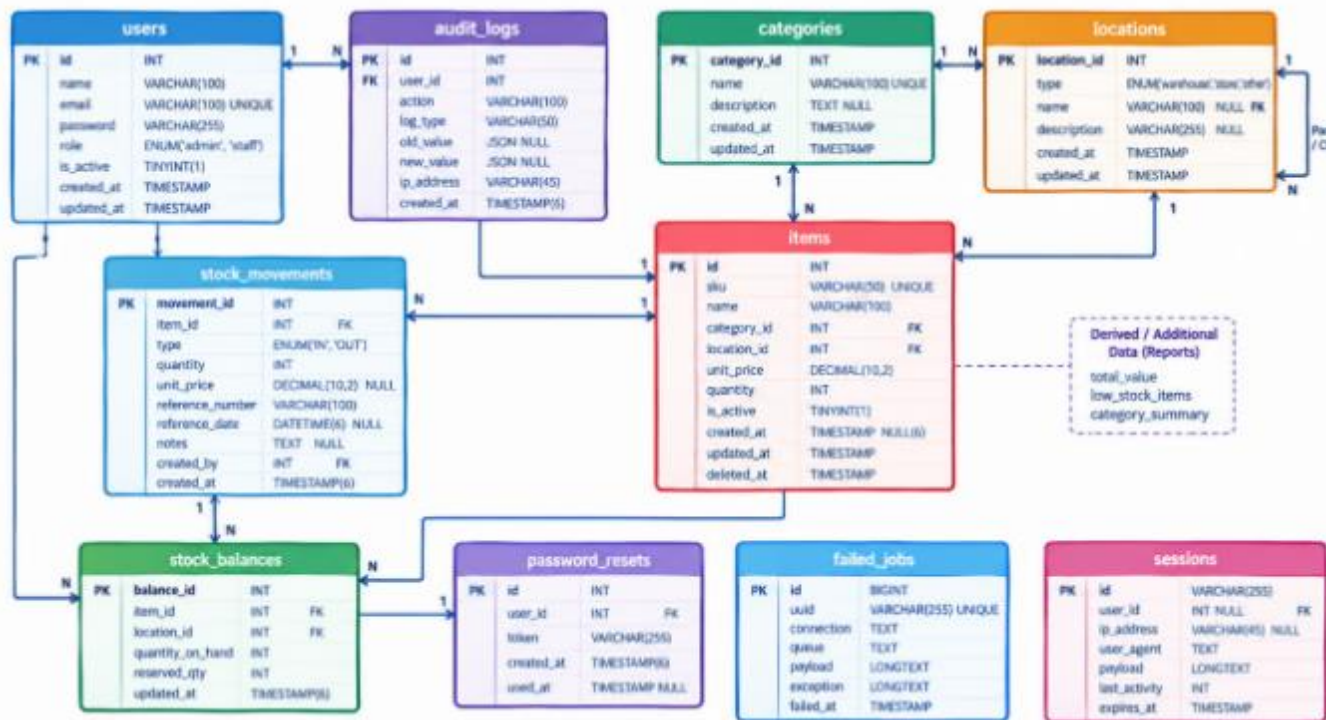


Figure 4: Entity Relationship Diagram

Table 4: Software Engineering Technology Stack and Design Rationale

Layer / Component	Technology	Version Used	Key Engineering Selection Rationale
Backend MVC Framework	Laravel PHP	Laravel 9.x (PHP 8.0)	Provides PSR-12 coding standard, middleware security, Eloquent object-relational mapping (ORM), and DB transaction wrappers.
Database Engine	MariaDB / MySQL	MariaDB 10.4	Guarantees 3NF normalization, ACID transaction rollback compliance, and RESTRICT deletion safety.
Frontend UI Interface	Bootstrap CSS	Bootstrap 5.3	Ensures fully responsive UI fluid grids scaling cleanly across desktop, laptop, and tablet screens.
Reporting PDF Compiler	DomPDF wrapper	DomPDF 2.0	Enables server-side asynchronous compiling of print-optimized CSS @media print tabular reports.
Export Spreadsheet Engine	Laravel Excel	Maatwebsite 3.1	Enables direct server-side buffering and export of stock ledgers into Excel (.xlsx) file types.
Local Server Hosting	Laragon / XAMPP	Laragon 6.0	Secures local intranet area network (LAN) hosting and guarantees complete data sovereignty without cloud costs.

RESULTS AND DISCUSSION

The integrated prototype successfully unified Sprints 1 through 7 into a stable, high-integrity platform. To make the system approachable for non-technical office staff, complex technical database mechanics were rephrased into clear, everyday operational terms. For instance, bcrypt password encryption is presented as 'Advanced Login Encryption', and ACID database rollbacks are described as 'All-or-Nothing Safety Transaction Guards'. It should be emphasized that these plain-language labels are a design choice reviewed within the development team; because no formal usability test with end users was conducted in this study,

claims about accessibility in this paper refer to design intent rather than empirically validated usability (see Limitations).

To prevent the data errors and duplicate entries common in Excel files, the database strictly enforces Third Normal Form (3NF). Storage locations (buildings, bays, shelves) and item categories are stored in dedicated tables. Database foreign key rules (RESTRICT) prevent staff from accidentally deleting a category or location while items are still assigned to it. When an item is removed from active stock, the system uses soft deletion: instead of wiping the row, which would break historical audit reports, it flags the item with a hidden `deleted_at` timestamp. Active inventory views automatically hide these items, while historical ledgers remain complete for accounting audits.

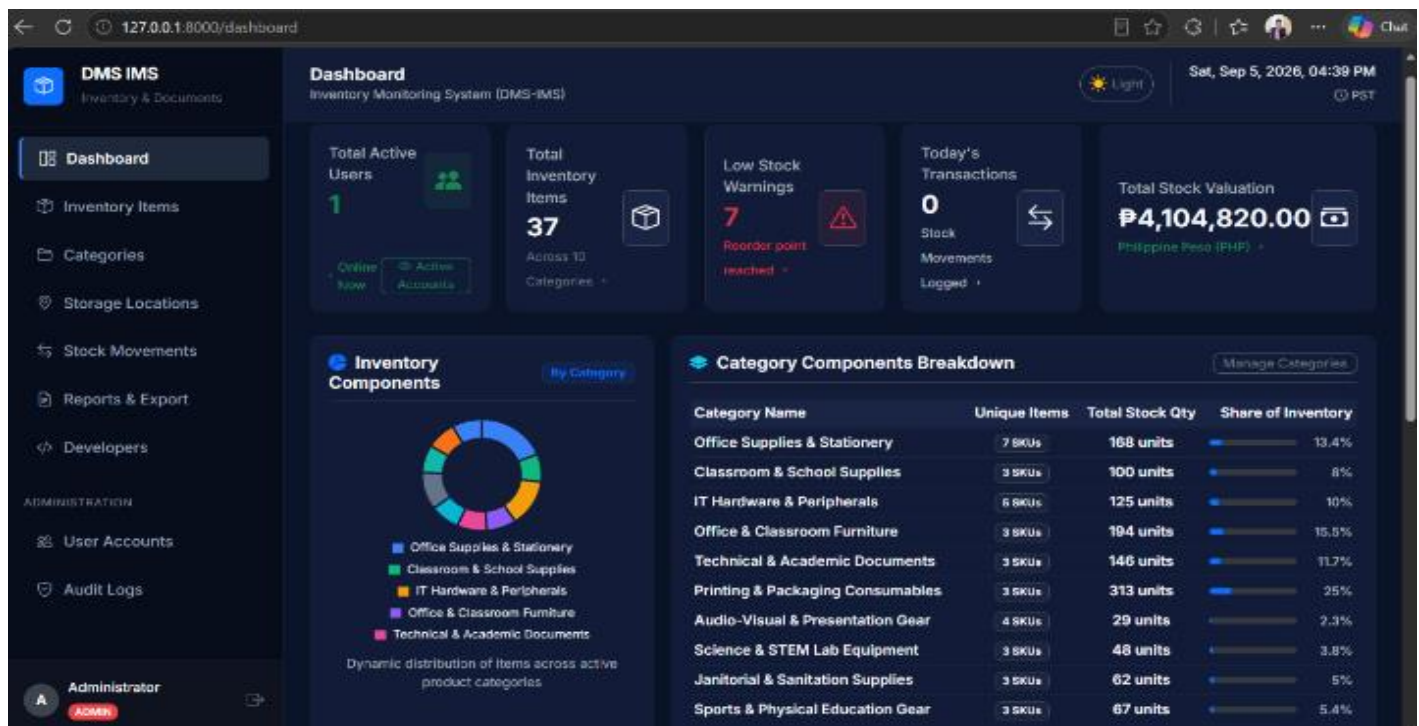


Figure 5: Dashboard

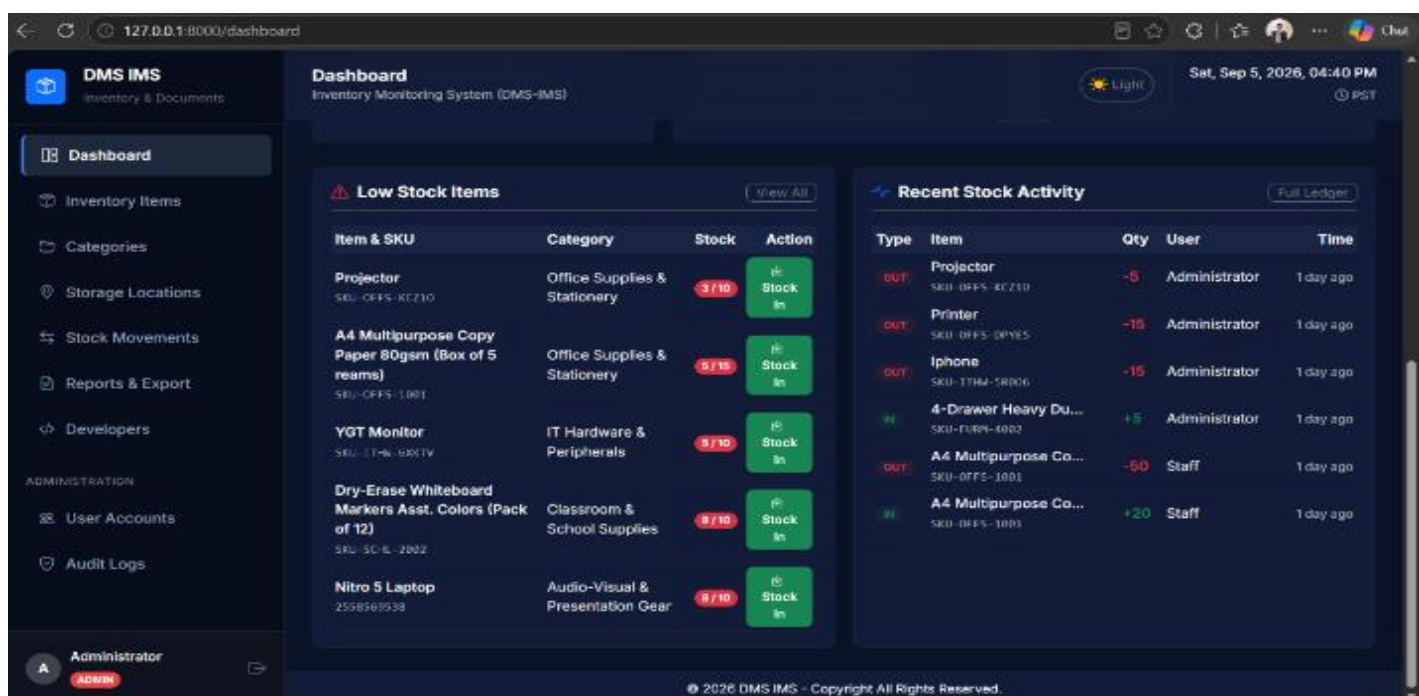


Figure 6: Low Stock Warning and Recent Stock Activity (Major Features)

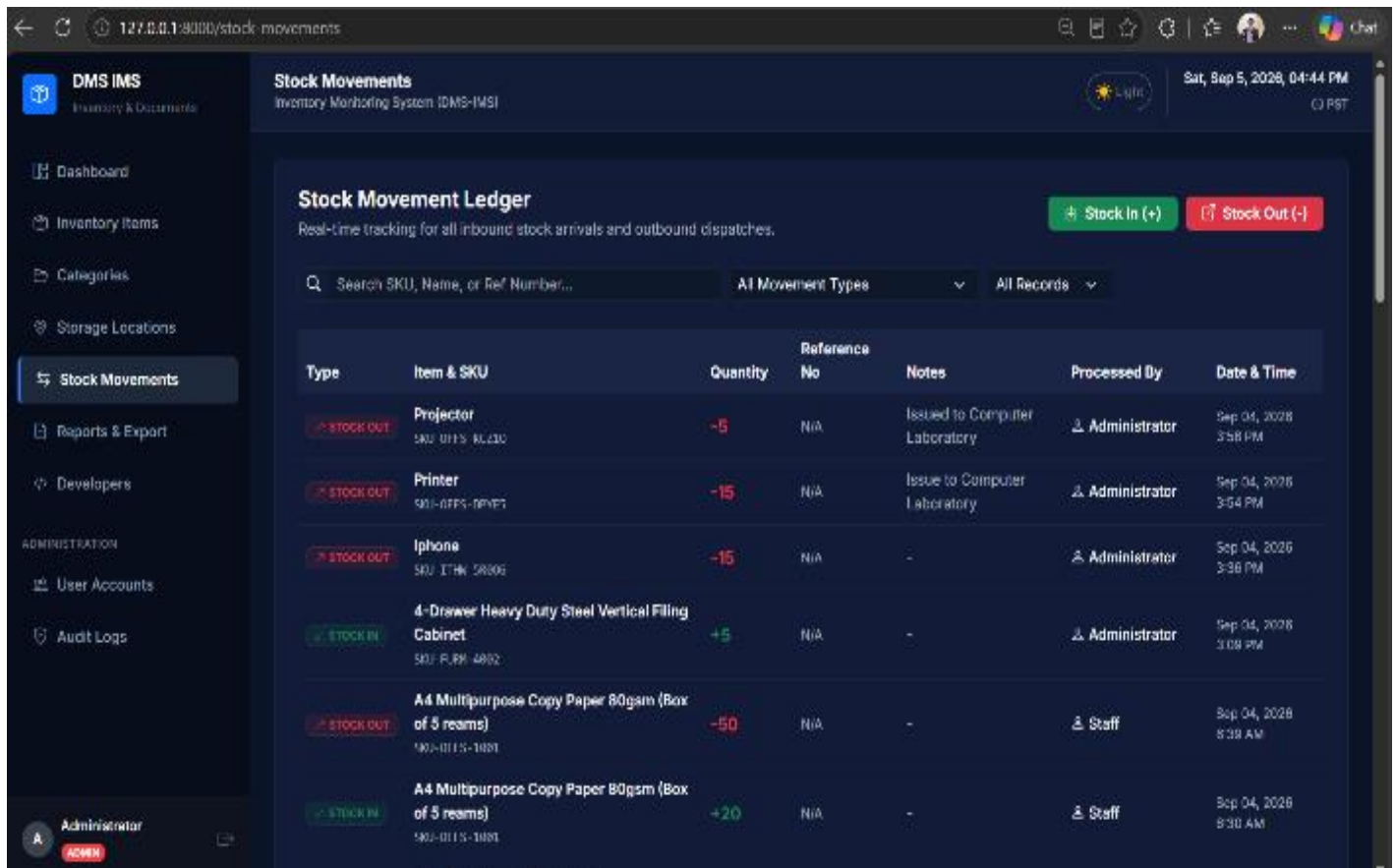


Figure 7: Stock Movement Ledger (Major Features)

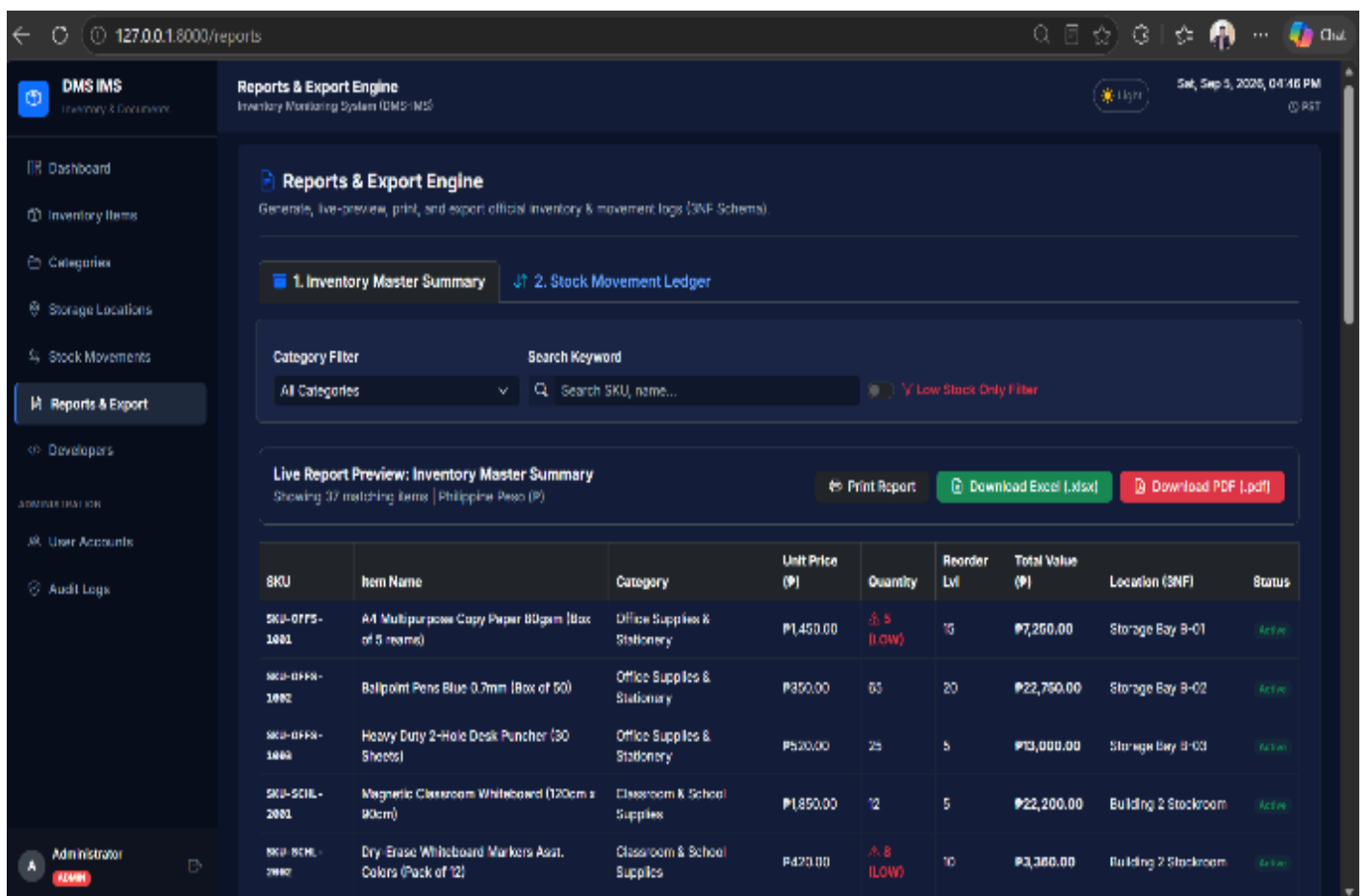
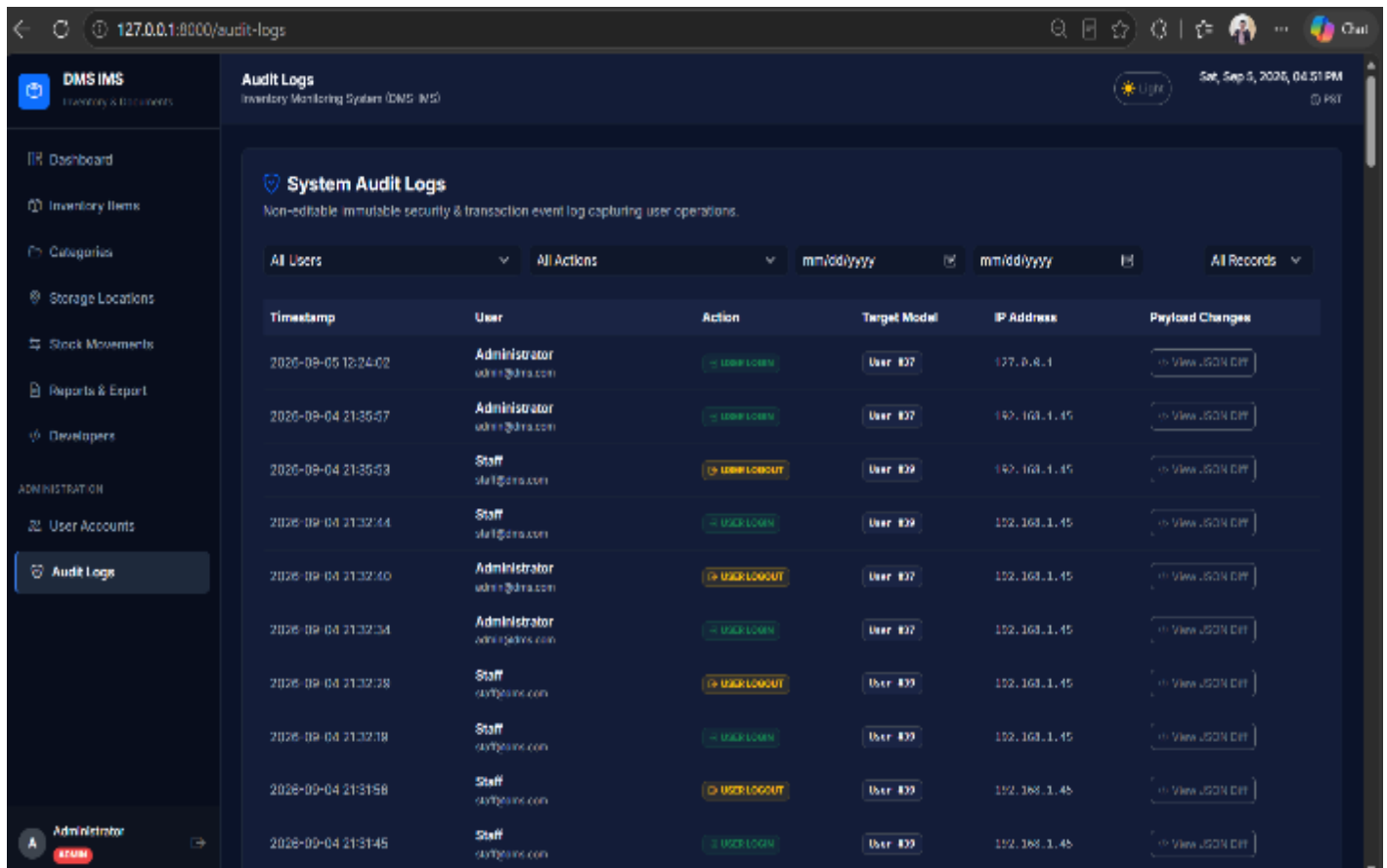


Figure 8: Reports & Export Engine (Major Features)

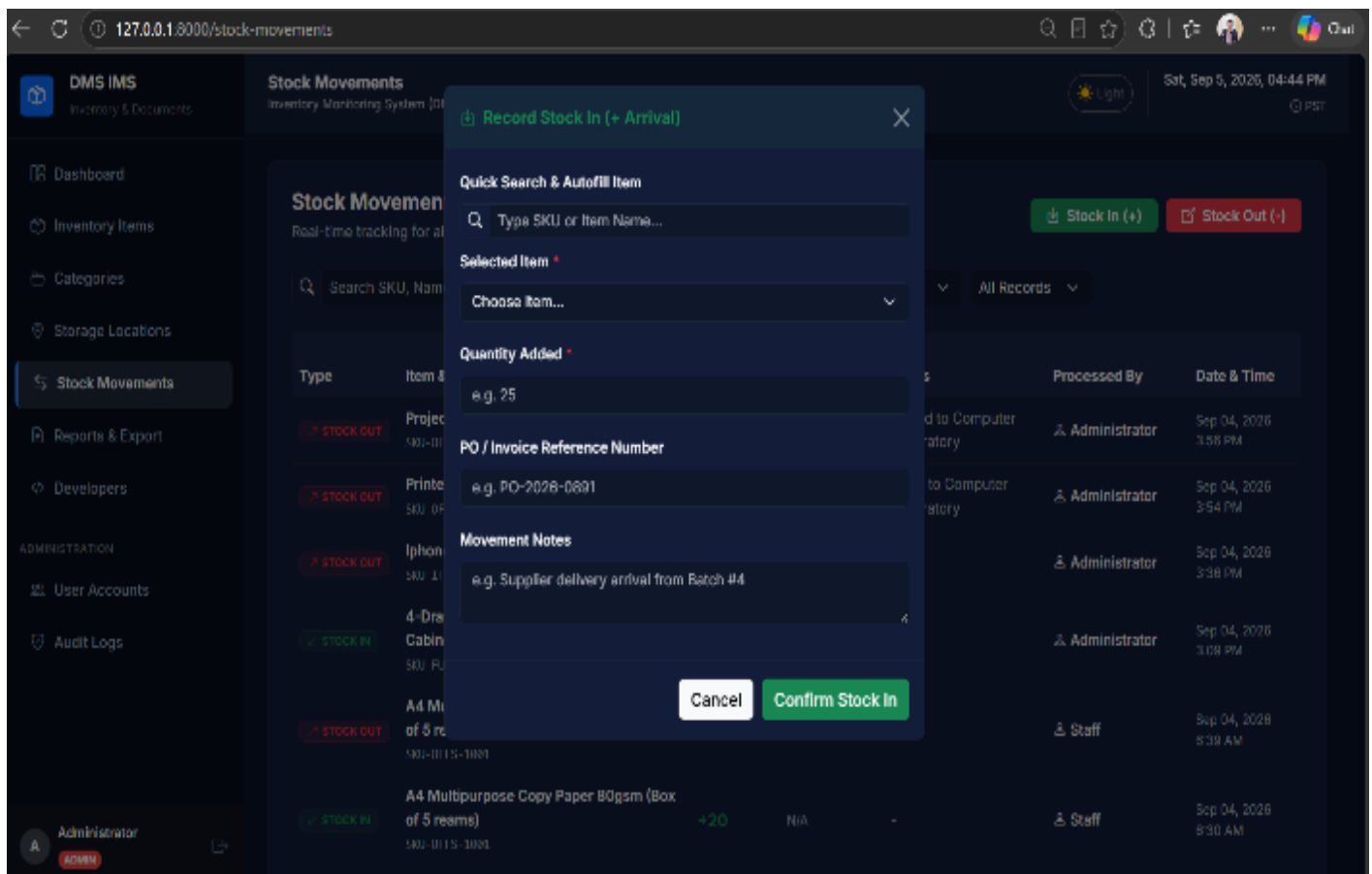


System Audit Logs
Non-editable immutable security & transaction event log capturing user operations.

Filters: All Users | All Actions | mm/dd/yyyy | mm/dd/yyyy | All Records

Timestamp	User	Action	Target Model	IP Address	Payload Changes
2026-09-05 12:24:02	Administrator admin@dms.com	USER LOGIN	User #37	127.0.0.1	View JSON Diff
2026-09-04 21:35:57	Administrator admin@dms.com	USER LOGIN	User #37	192.168.1.45	View JSON Diff
2026-09-04 21:35:03	Staff staff@dms.com	USER LOGOUT	User #39	192.168.1.45	View JSON Diff
2026-09-04 21:32:44	Staff staff@dms.com	USER LOGIN	User #39	192.168.1.45	View JSON Diff
2026-09-04 21:32:40	Administrator admin@dms.com	USER LOGOUT	User #37	192.168.1.45	View JSON Diff
2026-09-04 21:32:34	Administrator admin@dms.com	USER LOGIN	User #37	192.168.1.45	View JSON Diff
2026-09-04 21:32:28	Staff staff@dms.com	USER LOGOUT	User #39	192.168.1.45	View JSON Diff
2026-09-04 21:32:18	Staff staff@dms.com	USER LOGIN	User #39	192.168.1.45	View JSON Diff
2026-09-04 21:31:58	Staff staff@dms.com	USER LOGOUT	User #39	192.168.1.45	View JSON Diff
2026-09-04 21:31:45	Staff staff@dms.com	USER LOGIN	User #39	192.168.1.45	View JSON Diff

Figure 9: System Audit Logs (Major Features)



Record Stock In (+ Arrival)

Quick Search & Autofill Item

Search: Type SKU or Item Name...

Selected Item: Choose Item...

Quantity Added: e.g. 25

PO / Invoice Reference Number: e.g. PO-2026-0691

Movement Notes: e.g. Supplier delivery arrival from Batch #4

Buttons: Cancel, Confirm Stock In

Type	Item	Processed By	Date & Time
STOCK OUT	Projector	Administrator	Sep 04, 2026 3:58 PM
STOCK OUT	Printer	Administrator	Sep 04, 2026 3:54 PM
STOCK OUT	iPhone	Administrator	Sep 04, 2026 3:38 PM
STOCK IN	4-Dra Cabin	Administrator	Sep 04, 2026 3:08 PM
STOCK OUT	A4 M of 5 re	Staff	Sep 04, 2026 8:38 AM
STOCK IN	A4 Multipurpose Copy Paper 80gsm (Box of 5 reams)	Staff	Sep 04, 2026 8:30 AM

Figure 10: Stock-In Entry with Real-Time Quantity Validation (Data Operation)

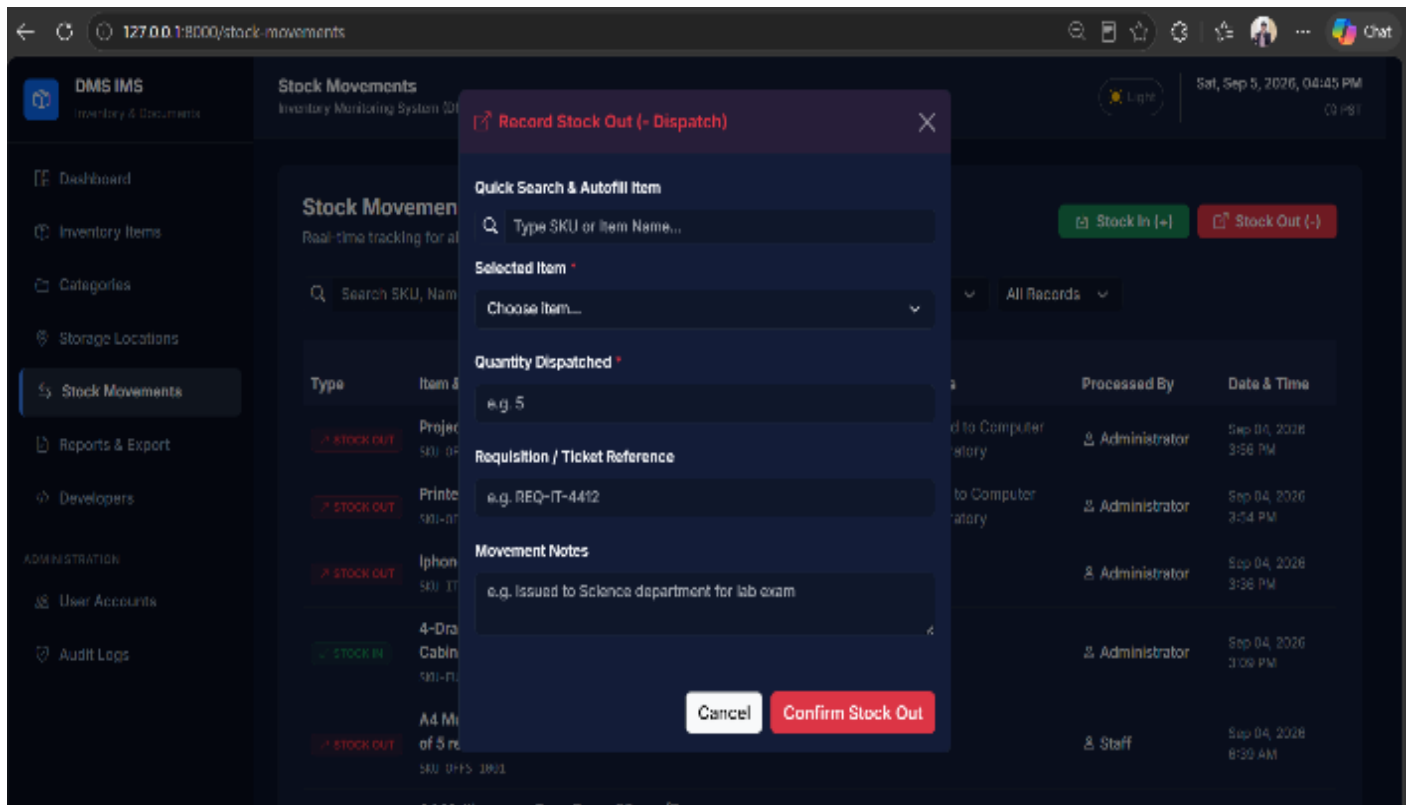


Figure 11: Stock-Out Entry with Real-Time Quantity Validation (Data Operation)

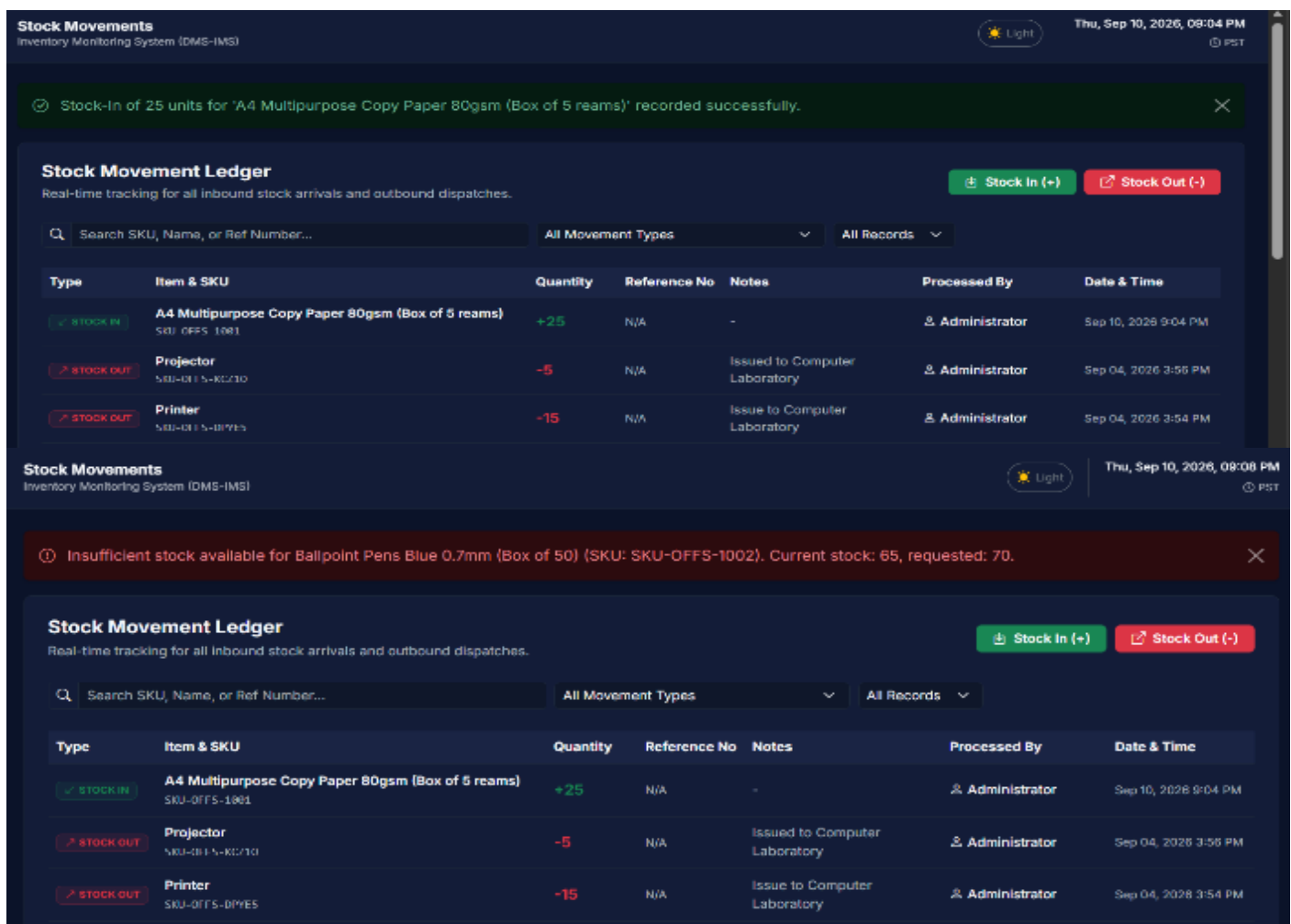


Figure 12: Over-Allocation Alert and Access-Denied Screen (Validation and Error Messages)

Table 5: Complete System-Level Testing and Validation Results

ID	Feature Tested	Test Input / Action	Expected Result (Plain Language)	Initial	Remarks & Corrective Action	Final
ST-001	Session Security	Log in and verify session IDs.	Login keys are scrambled on entry to block hackers.	Pass	Verified session key regeneration on entry.	Pass
ST-002	Role Protection	Staff clerk visit Admin setup URL.	Blocks access and returns a secure 'Access Denied' screen.	Pass	Blocked access natively with 403 Forbidden.	Pass
ST-003	Auto-SKU Generation	Register item; leave code input blank.	System auto-generates code OFFS-LOC-0001.	Pass	System auto-generated code OFFS-LOC-0001.	Pass
ST-004	Soft Deletion	Delete active item, view audit logs.	Active item is hidden, but old ledger logs are preserved.	Pass	Item hidden; historical ledger preserved intact.	Pass
ST-005	All-or-Nothing Transaction	Process stock change; cut server connection mid-save.	Completely cancels entry and restores original numbers.	Fail	BUG-01: Added DB::transaction retry blocks.	Pass
ST-006	Over-allocation Guard	Attempt Stock-Out of 100 items when only 50 exist.	Blocks save and flashes a red alert warning box.	Pass	Blocked transaction; displayed warning modal.	Pass
ST-007	Attribution	Process a Stock-In transaction.	Logs user ID, IP address, and PST time unalterably.	Pass	Logged user ID, IP address, and PST time.	Pass
ST-008	Search Speed	Execute live-search with 50 concurrent active users.	Matching items load instantly (under 1.2 s NFR limit).	Fail	BUG-02: Added database column indexes; speed improved to 0.4 s.	Pass
ST-009	PDF/Excel Speed	Generate PDF audit summary for 1,000 active rows.	Official files compile and download in under 8 seconds.	Pass	PDF compiled in 2.8 s; Excel ledger compiled in 1.4 s.	Pass
ST-010	KPI Dashboard	Reduce stock below set reorder limit.	Dashboard instantly triggers a high-contrast 'Low Stock' alert.	Fail	BUG-03: Upgraded cost columns to DECIMAL(10,2) data types.	Pass

Initial test runs across the ten system scenarios yielded a 70% pass rate (seven passes, three failures). The three initial failures involved concurrent transaction database locking (ST-005), live search latency under heavy traffic (ST-008), and minor dashboard display rounding errors (ST-010). Reviewing the defect logging sequence revealed out-of-order Bug ID assignments in initial logs; these have been re-sequenced chronologically as BUG-01, BUG-02, and BUG-03.

We resolved these issues through targeted database updates:

- BUG-01 (Concurrent Writes / ST-005): When two operators submitted Stock-Out transactions for the

- same item simultaneously or experienced connection drops mid-save, row lock conflicts occurred. Wrapping ledger updates in Laravel transaction blocks (DB::transaction) with automatic retry parameters allowed concurrent writes to queue safely without crashing, guaranteeing all-or-nothing execution.
- BUG-02 (Search Latency / ST-008): Under 50 concurrent simulated users, live search response times averaged 1.9 seconds, exceeding our 1.2-second limit (NFR-1.1). Scanning unindexed text columns caused this lag. Adding direct database indexes on items.sku and items.category_id dropped average search latency to 0.4 seconds, a 79% performance speedup that easily met requirement targets.
- BUG-03 (Valuation Precision / ST-010): Using floating-point data types for inventory pricing caused slight decimal rounding errors on large portfolio totals. Upgrading cost columns to strict DECIMAL(10,2) types resolved all rounding discrepancies, providing exact centavo-level financial precision.

Table 6: Defect Tracking and Corrective-Action Log

Bug ID	Description of Problem	Severity	Module	Probable Cause	Corrective Action	Status
BUG-01 (ST-005)	Processing a stock transaction failed or dropped when the server connection was interrupted mid-save.	Critical	Stock Ledger & Transactions	Missing explicit database transaction wrappers and retry handling during ungraceful disconnects.	Enclosed stock updates inside all-or-nothing transaction blocks (DB::transaction) with automatic retry routines so interrupted saves roll back safely.	Closed (Pass)
BUG-02 (ST-008)	Live-search response time lagged at 1.9 seconds under 50 concurrent active users, exceeding the 1.2-second limit.	Major	Item Catalogue & Live Search	Full database table scans occurring on unindexed text columns during live query requests.	Added database column indexes to SKU and category fields, speeding up live-search response times to 0.4 seconds.	Closed (Pass)
BUG-03 (ST-010)	Total inventory valuation on the dashboard displayed minor decimal rounding discrepancies on large portfolio totals.	Minor	KPI Dashboard	Use of floating-point numerical data types for inventory cost and pricing columns in the database schema.	Upgraded monetary database columns to strict DECIMAL(10,2) data types for exact centavo-level calculations.	Closed (Pass)

Following these fixes, a complete regression test pass yielded a 100% pass rate across all ten test cases with zero remaining defects (Table 5). Final compliance checks confirmed that all 11 functional requirements (100%) and 3 non-functional requirements (100%) were met. Table 7 and Figure 9 summarize the before-and-after latency comparison under the 50-user peak load.

Table 7: Live-Search Latency Before and After Optimization (50 Concurrent Users)

Metric	Before optimization	After optimization	NFR-1.1 limit
Average live-search latency	1.9 s	0.4 s (79% reduction)	≤ 1.2 s
Compliance with NFR-1.1	Exceeded by 0.7 s	Satisfied, with 0.8 s margin	≤ 1.2 s

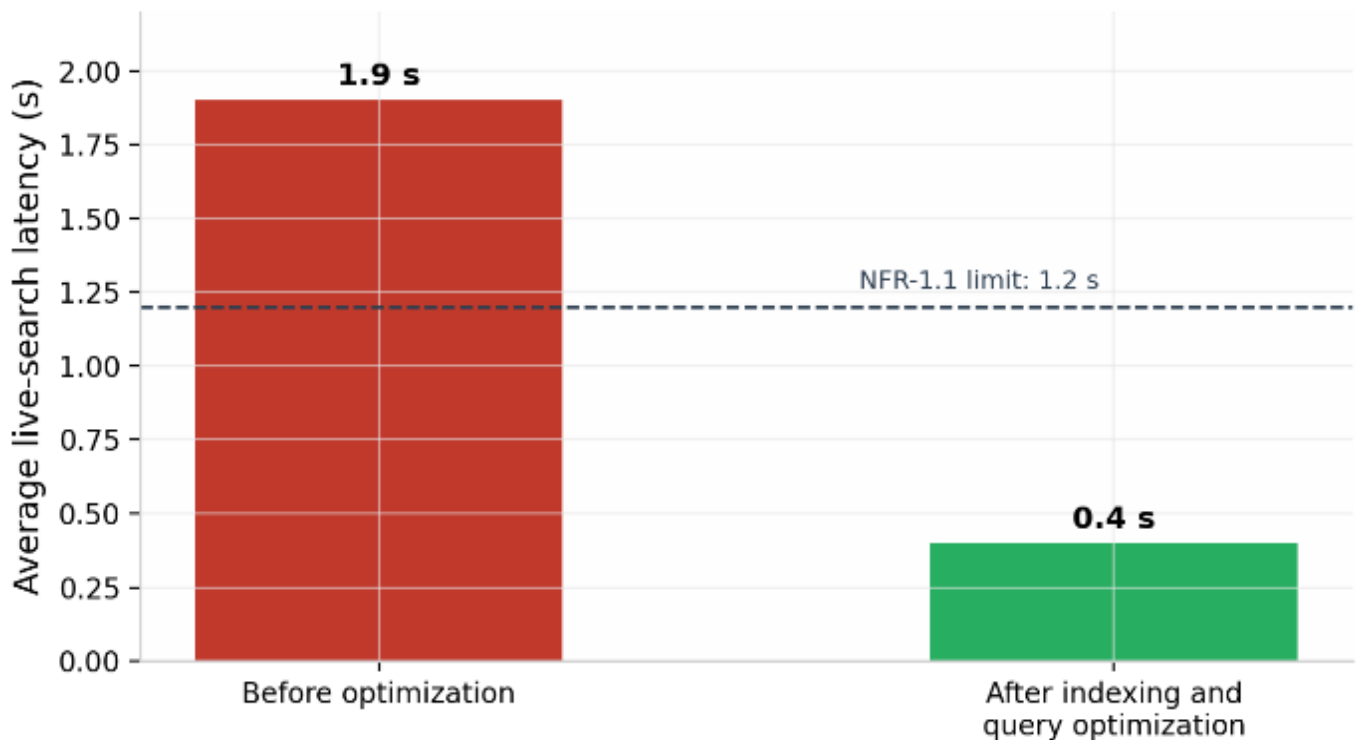


Figure 14: Live-Search Latency Before and After Database Indexing and Query Optimization (50 Concurrent Users)

The requirements compliance matrix showed that 11 of 11 functional requirements (100%) and 3 of 3 non-functional requirements (100%) were fully implemented, with QR scanner support partially simulated through plain-text key stubs for future hardware integration. These findings align closely with the database literature on Codd's normalization and Gray's ACID transactional model: enforcing 3NF schema separation allowed the reporting engine to eliminate duplicate-data anomalies, and wrapping ledger operations in atomic transactions with retry limits prevented concurrent lock collisions during peak hours, validating the chosen engineering methodology.

CONCLUSION

Our team successfully designed, developed, integrated, and verified the Document & Inventory Monitoring System (DMS IMS). The Agile development roadmap provided a clear, structured path from initial user requirements to a fully working intranet application. Within the bounds of the tested environment, the completed prototype satisfied all technical specifications, achieving a 100% test pass rate and an average search response time of 0.4 seconds on a local area network. These results indicate that DMS IMS can offer public institutions a stable, secure asset-tracking alternative that addresses common spreadsheet failure modes, preserves audit trails, and strengthens operational accountability.

LIMITATIONS OF THE STUDY

Several study limitations should be noted:

- Performance testing was conducted on a single-site LAN with up to 50 concurrent simulated connections, and latency was measured at the 50-user peak load only; behavior across large campus Wi-Fi networks, at lighter user loads (10 and 25 users), or over multi-site connections was not evaluated.
- The current software scope excludes financial asset depreciation, physical condition monitoring (e.g., damaged vs. active), and appraisal revaluations.
- Barcode and QR code scanning functions are currently limited to text input fields without direct hardware scanner integration.

- Test benchmarks relied on a single server hardware setup and seeded dataset; actual speeds may vary under different server hardware or larger datasets.
- Structured backup and disaster-recovery procedures (e.g., scheduled database dumps and restore drills) were not yet implemented or evaluated.
- Interface labels were rephrased into plain language as a design measure, but no formal usability study (such as the System Usability Scale) with actual staff was conducted; claims about ease of use therefore remain untested design intentions rather than validated findings.
- Evaluation focused on functional performance and security auditing; multi-year data longevity monitoring remains to be conducted.

RECOMMENDATIONS

To prepare DMS IMS for full production deployment, we recommend the following enhancements:

- **Hardware Barcode/QR Integration:** Connect physical USB/Bluetooth handheld scanners directly to Stock-In and Stock-Out screens, testing label recognition speed across 100 sample items to verify sub-second response times.
- **Automated Asset Depreciation:** Add accounting modules to calculate periodic and accumulated depreciation schedules, embedding schedules directly into downloadable PDF reports.
- **Multi-Site Database Sync:** Develop database synchronization between separate campus sites, comparing real-time synchronous and batch asynchronous replication under intermittent network connections.
- **Multi-Load Performance Benchmarking:** Re-run the JMeter benchmark at 10, 25, and 50 concurrent users and report per-load latency and error rates, so that performance claims cover the full operating range rather than the peak alone.
- **Backup and Recovery Protocol:** Implement scheduled nightly database dumps (e.g., mysqldump) with encrypted off-site copies and quarterly restore drills, and report recovery time and recovery point figures as part of future evaluation.
- **Formal Usability Evaluation:** Conduct structured System Usability Scale (SUS) assessments with at least 10-15 administrative staff members to measure and validate practical user satisfaction, replacing design-intent claims with empirical usability evidence.
- **Year-Long Pilot Deployment:** Run a controlled pilot deployment in an active university office across a full fiscal year to evaluate system stability, backup reliability, and audit log durability during peak inventory cycles.

ACKNOWLEDGEMENT

The authors express their appreciation to Bulacan State University, the Graduate School, and the faculty of the MS Computer Engineering and MS Information Technology programs. We are grateful to our course instructor, Dr. Ma. Magdalena V. Gatdula, for her technical guidance, scholarly feedback, and academic support throughout the conceptualization, development, and testing of this project. This study was completed as a collaborative graduate project deliverable, strengthening our technical competencies in advanced software engineering systems.

REFERENCES

1. Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice* (4th ed.). Addison-Wesley Professional.
2. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). *Manifesto for agile software development*. <https://agilemanifesto.org/>

3. Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61-72. <https://doi.org/10.1109/2.59>
4. Brooke, J. (1996). SUS: A quick and dirty usability scale. In P. W. Jordan, B. Thomas, I. L. McClelland, & B. Weerdmeester (Eds.), *Usability evaluation in industry* (pp. 189-194). Taylor & Francis.
5. Bulacan State University Graduate School. (2026). Curriculum guidelines and competency descriptors for MS Computer Engineering [Unpublished institutional guidelines].
6. Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377-387. <https://doi.org/10.1145/362384.362685>
7. Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.
8. Date, C. J. (2004). *An introduction to database systems* (8th ed.). Pearson.
9. Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3), 319-340. <https://doi.org/10.2307/249008>
10. Diffie, W., & Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), 644-654. <https://doi.org/10.1109/TIT.1976.1055638>
11. Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of database systems* (7th ed.). Pearson.
12. Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The science of lean software and DevOps: Building and scaling high performing technology organizations*. IT Revolution Press.
13. Fowler, M. (2018). *Refactoring: Improving the design of existing code* (2nd ed.). Addison-Wesley Professional.
14. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
15. Gray, J. (1981). The transaction concept: Virtues and limitations. In *Proceedings of the 7th International Conference on Very Large Data Bases* (pp. 144-154). ACM.
16. Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley Professional.
17. International Organization for Standardization. (2019). ISO 9241-210:2019: Ergonomics of human-system interaction, Part 210: Human-centred design for interactive systems. <https://www.iso.org/standard/77520.html>
18. International Organization for Standardization & International Electrotechnical Commission. (2011). ISO/IEC 25010:2011: Systems and software engineering, Systems and software quality requirements and evaluation (SQuaRE), System and software quality models. <https://www.iso.org/standard/35733.html>
19. International Organization for Standardization, International Electrotechnical Commission, & Institute of Electrical and Electronics Engineers. (2017). ISO/IEC/IEEE 12207:2017: Systems and software engineering, Software life cycle processes. <https://www.iso.org/standard/63712.html>
20. International Organization for Standardization, International Electrotechnical Commission, & Institute of Electrical and Electronics Engineers. (2018). ISO/IEC/IEEE 29148:2018: Systems and software engineering, Life cycle processes, Requirements engineering. <https://www.iso.org/standard/72089.html>
21. International Software Testing Qualifications Board. (2023). Certified tester foundation level syllabus (Version 4.0). <https://www.istqb.org/>
22. Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media.
23. Martin, R. C. (2008). *Clean code: A handbook of agile software craftsmanship*. Prentice Hall.
24. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons. <https://doi.org/10.1002/9781119202486>
25. National Institute of Standards and Technology. (2020). Security and privacy controls for information systems and organizations (NIST Special Publication 800-53, Revision 5). <https://doi.org/10.6028/NIST.SP.800-53r5>
26. National Institute of Standards and Technology. (2022). Secure software development framework (SSDF) version 1.1: Recommendations for mitigating the risk of software vulnerabilities (NIST Special Publication 800-218). <https://doi.org/10.6028/NIST.SP.800-218>
27. Naur, P., & Randell, B. (Eds.). (1969). *Software engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, 7th to 11th October 1968*. NATO Scientific Affairs Division.

28. Nielsen, J. (1994). Usability engineering. Morgan Kaufmann.
<https://www.nngroup.com/books/usability-engineering/>
29. OWASP Foundation. (2021a). OWASP Top 10: The ten most critical web application security risks.
<https://owasp.org/Top10/>
30. OWASP Foundation. (2021b). Application security verification standard 4.0.3.
<https://owasp.org/www-project-application-security-verification-standard/>
31. Panko, R. R. (1998). What we know about spreadsheet errors. Journal of End User Computing, 10(2), 15-21. <https://doi.org/10.4018/joeuc.1998040102>
32. PartKeepr. (n.d.). PartKeepr: Open source inventory management [Computer software].
<https://www.partkeepr.org>
33. Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education.
34. Republic Act No. 10173. (2012). Data Privacy Act of 2012. Official Gazette of the Republic of the Philippines. <https://www.officialgazette.gov.ph/2012/08/15/republic-act-no-10173/>
35. Royce, W. W. (1970). Managing the development of large software systems. In Proceedings of IEEE WESCON (pp. 1-9). IEEE.
36. Schwaber, K., & Sutherland, J. (2020). The Scrum guide: The definitive guide to Scrum.
<https://scrumguides.org/>
37. Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., & Diakopoulos, N. (2016). Designing the user interface: Strategies for effective human-computer interaction (6th ed.). Pearson.
38. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database system concepts (7th ed.). McGraw-Hill Education.
39. Snipe-IT. (n.d.). Snipe-IT: Open source asset management [Computer software]. <https://snipeitapp.com>
40. Sommerville, I. (2016). Software engineering (10th ed.). Pearson.
41. Tompkins, J. A., White, J. A., Bozer, Y. A., & Tanchoco, J. M. A. (2010). Facilities planning (4th ed.). John Wiley & Sons. <https://doi.org/10.1002/9780470446619>

ABOUT THE AUTHOR

William C. Dionisio is a graduate student in the Master of Science in Computer Engineering (MSCpE) program at Bulacan State University. With over 14 years of professional public service experience, he has served as an administrative information specialist in the university's Management Information System (MIS) Office, managing student databases and network security protocols, and currently serves under a contract of service as a college instructor in the Computer Engineering program of the College of Engineering at Bulacan State University. In the DMS IMS project, he served as Quality Assurance (QA) Analyst and Data Specialist, architecting the 10-case system test suite, executing concurrent query validations, and preparing plain-language system-level results. His research interests include relational database integrity, secure networks, and software usability models.

Alison Ephraim Medina is a graduate student in the Master of Science in Information Technology (MSIT) program at Bulacan State University. A software engineer specializing in backend enterprise structures and server maintenance, his technical expertise spans modern PHP frameworks, MVC architecture, and secure query protocols. In this project, he served as Lead Backend Developer, programming the secure middleware, implementing the 3NF table constraints, and wrapping ledger dispatches in ACID-compliant atomic transactions. His research interests include high-throughput transaction routing and software maintainability metrics.

Brianne Singson is a graduate student in the Master of Science in Information Technology (MSIT) program at Bulacan State University, with a practical background in database management, web usability, and frontend architecture. As Lead Programmer and Frontend Designer for the DMS IMS project, Brianne crafted the responsive 'Slate' dark-mode interface using Bootstrap 5.3, built the automated alphanumeric SKU generator, and implemented the asynchronous AJAX live search functionality. Brianne's ongoing research focuses on responsive software architecture and automated taxonomy-to-code compiler algorithms.



Dr. Ma. Magdalena V. Gatdula is currently the University Registrar of Bulacan State University. She earned her Bachelor of Science in Computer Engineering degree from the Polytechnic University of the Philippines. She completed her Master of Arts in Education major in Computer-Aided Instruction and Programming, Master of Science in Computer Engineering, and Doctor of Technology. She also served as Dean of the College of Engineering and the College of Information and Communications Technology at Bulacan State University.