ISSN: 2395-1664 (ONLINE)
DOI: 10.21917/ijms.2026.0360

ROLE-BASED LEAVE MANAGEMENT SYSTEM USING SPRING BOOT AND RESTFUL ARCHITECTURE

Raghabendra Kumar Shah, Aman Verma, Pabitra Khatri, Pasupuleti Niranjan and B. Ganga Bhavani
Department of Computer Science and Engineering, Bonam Venkata Chalamayya Engineering College, India

Abstract

Traditional leave management processes in academic and corporate institutions continue to suffer from inefficiencies caused by paper-based workflows, weak validation, inconsistent leave balance tracking, and limited system integration. Although several web-based solutions exist, many lack transactional consistency, enforce minimal business rules, or fail to follow modular architectural principles. This paper presents the design and implementation of a role-based Employee Leave Management System developed using Spring Boot and RESTful architectural principles. The system follows a layered architecture separating controller, service, repository, and data transfer concerns to improve maintainability and extensibility. Core business rules—including non-overlapping leave validation, balance sufficiency checks, controlled status transitions, and role restricted operations—are enforced at the service layer with declarative transaction management ensuring atomicity and data consistency. A key design decision defers leave balance deduction until managerial approval rather than request submission, preventing balance inconsistencies for rejected requests. Functional validation confirms correct enforcement of business rules and transactional behavior across common and edge-case scenarios. The proposed system demonstrates how structured architectural design and transaction-aware workflows can address limitations observed in existing leave management solutions and provides a foundation for further enhancement toward production-ready HR system integration.

Keywords:

Spring Boot, REST API, Role-Based Access Control, Transactional Integrity, Business Rule Validation

1. INTRODUCTION

Human Resource Management (HRM) encompasses essential organizational functions including employee administration, attendance tracking, and payroll processing [1]. Within this domain, leave management represents a fundamental yet operationally complex process requiring coordination between employees, managers, and administrative systems [2]. Traditional approaches predominantly paper-based or spreadsheet-driven introduce procedural inefficiencies such as delayed approvals, error-prone balance calculations, and limited visibility into request status [3]. Conventional workflows require employees to submit handwritten applications manually routed through departmental hierarchies before HR departments record outcomes. This results in extended approval timelines, frequent synchronization errors, and absence of centralized real-time visibility [4]. Approved leaves often require separate manual communication to attendance and payroll departments, increasing reconciliation overhead and the likelihood of inconsistencies. Academic research has explored digital leave management solutions ranging from basic web-based systems to integrated HR platforms [5]. However, existing implementations reveal persistent architectural limitations. Many systems employ procedural designs without clear architectural layering, reducing maintainability as complexity increases. Transaction management is frequently inadequate, creating risks of partial updates when multi-step operations are not executed atomically [6]. Validation frameworks are often insufficient, permitting invalid submissions that require manual correction [7]. This paper presents a Leave Management System developed as an academic prototype demonstrating modern software architectural principles and transactional data handling. The system is implemented as a RESTful web service using Spring Boot 4.0.1 with Java 21, following a three-tier client–server architecture separating presentation, business logic, and data persistence concerns. The primary focus is leave request lifecycle management from submission through managerial approval with supporting modules for employee profiles, departmental organization, attendance recording, holiday calendars, and payroll data structures. The implementation contributes to academic understanding of enterprise Java application design through several aspects. First, it illustrates strict architectural layering using a Controller–Service–Repository pattern, demonstrating how separation of concerns improves maintainability. Second, comprehensive server-side validation enforces business rules including temporal constraints, overlapping request detection, and status transition enforcement. Third, transaction management using Spring's declarative annotations demonstrates atomic coordination of database operations during approval workflows. Fourth, the system demonstrates RESTful API design through twenty endpoints organized across eight functional controllers. The remainder of this paper is organized as follows. Section 2 reviews related work and identifies architectural gaps. Section 3 presents the system architecture and technology stack. Section 4 details the design and implementation of core modules with emphasis on validation and transaction management. Section 5 presents functional validation and qualitative analysis. Section 6 discusses limitations and future enhancements. Section 7 concludes the paper.

2. LITERATURE REVIEW

Digital leave management systems have been widely studied as representative workflow automation applications within Human Resource Management. This section reviews existing academic implementations and identifies recurring architectural and functional limitations relevant to leave processing systems.

2.1 WEB-BASED IMPLEMENTATIONS

Early academic efforts focused on replacing paper-based workflows with web forms and database-backed storage. Adamu [1] developed a PHP-based leave management system incorporating authentication and approval workflows, demonstrating improved processing speed over manual methods. However, the procedural design lacked service-layer abstraction, limiting maintainability and testability. Alade et al. [3] proposed

a similar PHP–MySQL system for public-sector use, successfully eliminating paperwork but exhibiting weak validation logic and limited transaction coordination for multistep operations.

These studies illustrate a common trend in early implementations: emphasis on user interface and basic CRUD functionality, with minimal attention to architectural layering or transactional integrity.

2.2 PLATFORM-SPECIFIC SOLUTIONS

Some research explored platform-specific solutions. Kaushik et al. [13] developed an Android-based student leave management application enabling mobile submission and approval notifications. While effective within its scope, the platform-specific design restricted accessibility and prevented integration with web-based administrative systems. Similar limitations were reported by Sapona et al. [9], where lack of cross-platform support constrained scalability and system interoperability.

2.3 FRAMEWORK-BASED IMPLEMENTATIONS

More recent studies adopted modern web frameworks. Birje et al. [6] implemented a leave management system using the MERN stack, demonstrating responsive frontend design and REST API integration. However, the study acknowledged limitations in transaction management, as MongoDB's eventual consistency model introduced integrity risks during concurrent balance updates. Explicit transaction boundaries for compound operations were absent.

Other works explored intelligent automation. Harshika et al. [2] proposed an AI-powered leave management ecosystem employing semantic analysis and probabilistic scheduling. While innovative, the system required extensive training data and introduced computational complexity unsuitable for smaller organizations. Additionally, transactional integrity during automated balance operations was not comprehensively addressed.

2.4 IDENTIFIED GAPS

Analysis of existing literature reveals several recurring gaps. First, many systems lack clear architectural layering, with business logic intermingled with controllers or data access code, reducing maintainability. Second, validation logic is often enforced only at the user interface level, allowing invalid requests to bypass constraints through direct API access. Third, transaction management remains insufficient approval workflows frequently update request status and balances through separate non-atomic operations, creating inconsistency risks during failures [8].

Overlapping leave detection is often absent or simplistic, failing to capture partial or encapsulated date conflicts. Additionally, inconsistent handling of leave balance deduction whether at submission or approval creates discrepancies when rejected requests permanently consume balances [10].

The system presented in this paper addresses these gaps through strict architectural separation using layered MVC design, comprehensive server-side validation including sophisticated overlap detection, explicit transaction boundaries coordinating multi-table updates, and deferred balance deduction executed atomically during approval.

3. SYSTEM ARCHITECTURE

The proposed Leave Management System adopts a three-tier client-server architecture separating presentation, application logic, and data persistence concerns.

4. THREE-TIER ARCHITECTURE

The Presentation Tier consists of a ReactJS-based web application providing interfaces for leave submission, request tracking, and managerial approval. Communication with the backend occurs exclusively through RESTful HTTP APIs, ensuring loose coupling between components.

The Application Tier is implemented using Spring Boot 4.0.1 and encapsulates all business logic, validation rules, and transaction coordination. Internally, it follows a layered Model–View-Controller pattern comprising Controller, Service, and Repository sub-layers. Controllers handle HTTP routing and JSON responses, Services implement domain logic and transactional operations, and Repositories abstract database access using Spring Data JPA interfaces.

The Data Tier employs MySQL relational database managed through Hibernate ORM, providing persistent storage with ACID transaction guarantees and referential integrity enforcement through foreign key constraints. The overall architecture is illustrated in Fig.1.

4.1 RESTFUL API DESIGN

The application exposes twenty REST endpoints across eight domain-specific controllers: AuthController (authentication), Employee- Controller (workforce profiles), DepartmentController (organizational structure), LeaveRequestController (leave lifecycle), AttendanceController (attendance tracking), PayrollController (payroll operations), and HolidayController (holiday calendars).

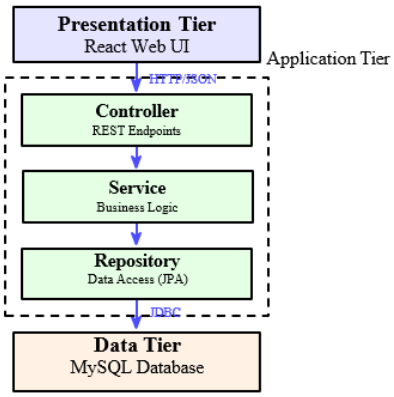

Fig.1. Three-tier architecture illustrating separation of concerns between presentation, application logic, and persistent storage.

REST endpoints follow resource-oriented naming with appropriate HTTP verbs: POST for creation (/api/leaves, /api/employees), GET for retrieval (/api/leaves/employee/{employeeId}), and PUT for updates (/api/leaves/{leaveId}/approve). JSON serves as the data interchange format. HTTP status codes communicate outcomes 200 OK for success, 201 Created for new resources, 400 Bad

Request for validation failures, 404 Not Found for missing resources. Data Transfer Objects decouple API contracts from persistence entities. The REST API endpoint organization is presented in Table.1.

4.2 DATA MODEL AND ENTITY RELATIONSHIPS

The data model comprises seven JPA entities with defined relationships. User stores authentication credentials (email, password) and role designation (EMPLOYEE, MANAGER). Employee extends user information with organizational profile details (name, email, designation, department, basic salary, status), maintaining one-to-one relationship with User. Department represents organizational structure (department name, description) with one-tomany relationship to Employee. LeaveRequest captures leave applications with foreign key to Employee, enumerated status (PENDING, APPROVED, REJECTED), and type (CASUAL, SICK, PAID).

Attendance records employee presence status by date. Payroll maintains monthly compensation records with leave deductions. Holiday stores organizational calendar entries.

Entity relationships employ JPA annotations (@OneToOne, @OneToMany, @ManyToOne) with proper cascade configurations. Foreign key constraints enforce referential integrity. Enumerated types prevent invalid data entry, demonstrating proper relational modeling supporting data integrity while remaining extensible. The entity relationship diagram is shown in Fig.2.

Table.1. RESTful endpoint organization across backend controllers

Controller	Endpoint	Method	Purpose
Auth	/api/auth/register	POST	User registration
Auth	/api/auth/login	POST	User login
Employee	/api/employees	POST	Create employee
Employee	/api/employees/{id}	GET	Getemployee
Department	/api/departments	POST	Create dept
Department	/api/departments	GET	List depts
Leave	/api/leaves	POST	Submit leave
Leave	/api/leaves/employee/{id}	GET	Get leaves
Leave	/api/leaves/{id}/approve	PUT	Approve leave
Leave	/api/leaves/{id}/reject	PUT	Reject leave
Attendance	/api/attendance	POST	Record attendance
Payroll	/api/payrolls/generate	POST	Generate payroll
Holiday	/api/holidays	POST	Addholiday
Holiday	/api/holidays	GET	Listholidays

4.3 TECHNOLOGY STACK SELECTION

Spring Boot 4.0.1 provides rapid development with embedded Tomcat, dependency injection, and auto-configuration. Spring Data JPA with Hibernate ORM simplifies object–relational mapping, eliminating manual SQL for standard CRUD operations. MySQL provides relational storage with ACID transaction support. Maven coordinates dependency management and build automation. The ReactJS frontend demonstrates API consumption patterns but is not treated as a research contribution emphasis remains on backend architecture and transactional behavior.

5. SYSTEM DESIGN AND IMPLEMENTATION

This section describes the design and implementation of core functional modules with emphasis on validation logic, transaction management, and integration patterns.

5.1 AUTHENTICATION AND ROLE MANAGEMENT

The authentication module implements basic user registration and login maintaining simplicity appropriate for academic projects. User registration via /api/auth/register validates email uniqueness before creating User entity. Login authentication via /api/auth/login validates credentials and returns AuthResponse containing user identifier, email, and role designation. The frontend maintains this information in browser localStorage, providing identifiers with subsequent requests. This stateless approach avoids server-side session complexity, though Spring Security integration represents

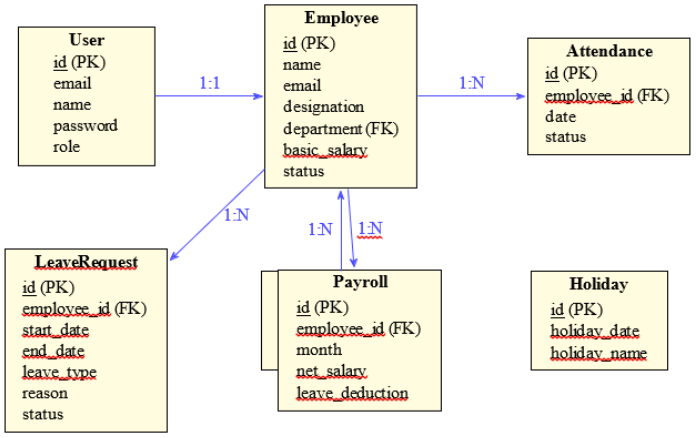

Fig.3. Entity-relationship diagram representing the normalized database schema and inter-entity dependencies

5.2 LEAVE REQUEST LIFECYCLE AND VALIDATION

Leave request management demonstrates comprehensive validation logic and transaction coordination. Leave application submission through /api/leaves POST endpoint initiates multistage validation at service layer boundaries. The complete lifecycle is illustrated in Fig.3.

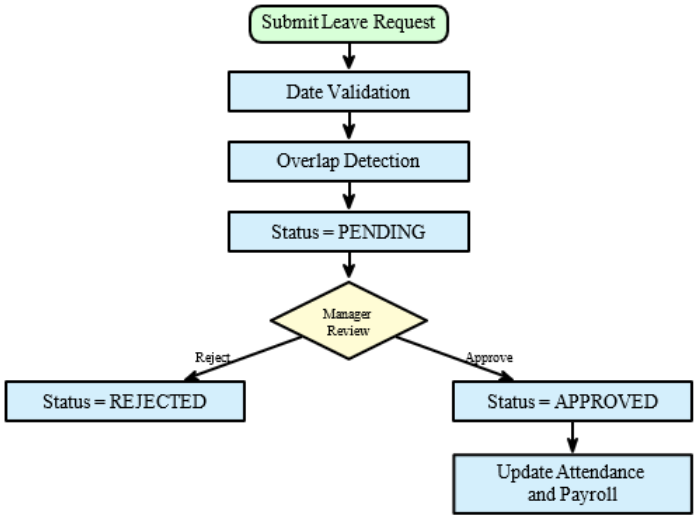

Fig.3. Lifecycle of leave request processing including validation and approval stages

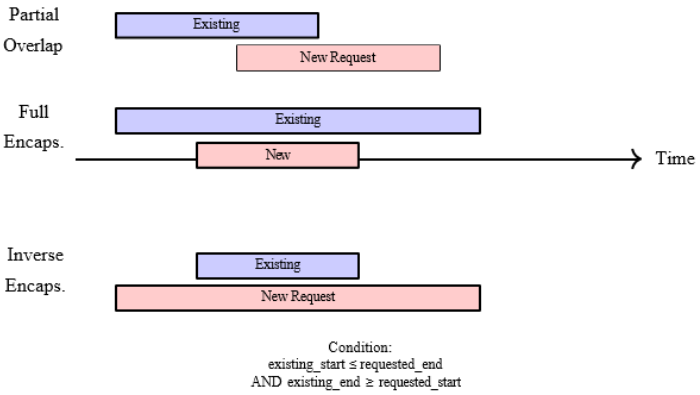

Fig.4. Visualization of date range intersection scenarios detected during leave validation

5.2.1 Comprehensive Validation Framework:

The service method implements layered validation: (1) Temporal validation verifies start date does not precede current date using LocalDate.now() comparison, preventing retroactive requests. (2) Chronological validation ensures end date equals or exceeds start date. (3) Leave type validation confirms type matches enumeration values (CASUAL, SICK, PAID). (4) Employee existence validation queries Employee repository confirming valid identifier.

5.2.2 Overlapping Leave Detection:

Overlapping leave detection executes database query identifying conflicts with existing requests. The repository employs JPQL query:

SELECT lr FROM LeaveRequest lr WHERE lr.employee.id=:employeeId AND lr.status IN ('PENDING', 'APPROVED') AND (lr.startDate <= :endDate AND lr.endDate >= :startDate)

This query retrieves leave requests matching three conditions: same employee, non-terminal status (excluding REJECTED), and date range intersection computed through boolean predicate. This predicate captures all overlap scenarios: partial overlap, complete encapsulation, and inverse encapsulation. Non-empty result triggers validation failure with descriptive error message. The overlap detection logic is visualized in Fig.4.

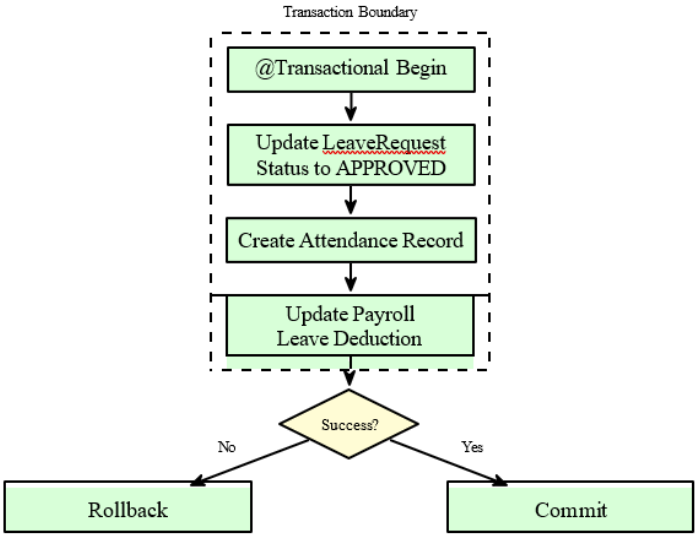

Fig.5. Atomic approval workflow ensuring consistency across leave, attendance, and payroll records.

5.2.3 Deferred Balance Deduction Design:

A critical design decision addresses when leave balances should decrease. Immediate deduction upon submission creates consistency issues when requests are rejected balances decrease but leave never occurred, requiring manual restoration. The implemented approach defers balance deduction until approval confirmation. Leave submission creates LeaveRequest entity with PENDING status without modifying balances. Only upon manager approval does the system execute balance deduction within the same transaction as status update.

5.2.4 Transaction-Coordinated Approval Workflow:

Leave approval through /api/leaves/{leaveId}/approve PUT endpoint demonstrates transaction management principles. The service method annotated with @Transactional executes multiple database operations atomically. Spring's transaction manager initiates database transaction before method execution. The method retrieves LeaveRequest entity, validates PENDING status, updates status to APPROVED, performs balance deduction, and saves modified entities.

The @Transactional annotation establishes transaction boundaries with READ_COMMITTED isolation level. If any operation fails status update, balance modification, or attendance record creation exception propagation triggers automatic rollback through Spring's transaction infrastructure. The database transaction aborts and all modifications revert, preventing partial updates. This demonstrates ACID property preservation: operations execute atomically (all succeed or all fail), maintaining consistency invariants (approved leaves always have corresponding balance deductions), with isolation preventing concurrent transactions from observing intermediate states. The transaction workflow is illustrated in Fig.5.

5.2.5 Leave Rejection and Retrieval:

Leave rejection through /api/leaves/{leaveId}/reject updates status to REJECTED without balance impact, since deferred deduction means balance never decreased. Leave retrieval

supports employee-specific history via /api/leaves/employee/{employeeId} GET for personal tracking and manager queues via /api/leaves GET with optional status filtering.

5.3 EMPLOYEE AND DEPARTMENT MANAGEMENT

The Employee module manages workforce master data through CRUD operations. Employee creation via /api/employees POST establishes one-to-one relationship with User entity and manyto-one association with Department, demonstrating proper JPA relationship modeling. The Department module provides organizational structure management via /api/departments endpoints. While current implementation maintains flat organizational structure, the design supports future enhancement to multilevel hierarchies.

5.4 ATTENDANCE, HOLIDAY, AND PAYROLL INTEGRATION FOUNDATIONS

5.4.1 Attendance Module:

The Attendance module implements basic record creation via /api/ attendance POST and retrieval via /api/attendance/employee/ {employeeId} GET. The design establishes integration touchpoint where leave approval could automatically generate attendance records marking approved dates as leave-type attendance. Current implementation provides data structures and CRUD operations without implementing automatic generation this integration represents identified future enhancement.

5.4.2 Holiday Calendar:

The Holiday module maintains organizational calendar through creation via /api/holidays POST and listing via /api/holidays GET. The implementation establishes foundation for intelligent leave day calculation where duration computation would exclude holidays from working day counts. Current implementation stores holiday data without active integration to leave validation logic. However, the modular design allows future integration with holidayaware leave deduction systems, addressing limitations noted in studies like Mantri et al. [10].

5.4.3 Payroll Foundation:

The Payroll module provides simplified data structures and basic operations. Payroll generation via /api/payrolls/generate POST initiates processing, while retrieval via /api/payrolls/employee/ {employeeId} GET provides historical access. The implementation establishes data model relationships enabling payroll calculations to query leave and attendance data. Current version maintains entity structure without implementing detailed calculation logic incorporating leave deductions this represents future work demonstrating comprehensive HR integration.

5.5 DATA TRANSFER OBJECTS AND EXCEPTION HANDLING

The implementation employs DTOs separating API contracts from database entities. Request DTOs (LeaveRequestDto, EmployeeRequest) define client-submitted data with Jakarta Validation annotations. Response DTOs (LeaveResponseDto, EmployeeResponse) control data exposure, excluding sensitive fields. Service methods convert between DTOs and entities, enabling independent evolution of API contracts and database schemas.

Exception handling employs custom exception classes extending RuntimeException. InvalidLeaveException signals validation failures. NotFoundException indicates missing entities. Controllers catch service layer exceptions, construct error response objects with HTTP status codes (400 for validation, 404 for not found, 500 for unexpected errors), descriptive messages, and timestamps. This centralized error handling ensures consistent API behavior.

5.6 IMPLEMENTATION SCOPE AND SIMPLIFICATIONS

Several simplifications distinguish this implementation from production systems. Password storage lacks BCrypt encryption. Spring Security integration is absent authentication relies on client-provided identifiers without server-side session validation. Leave balance tracking exists as data model structure with simplified implementation. Weekend and holiday exclusions in leave day calculation remain unimplemented. These simplifications focus implementation effort on demonstrating core architectural principles including layered design, validation frameworks, and transaction management rather than comprehensive feature coverage.

6. RESULTS AND PERFORMANCE EVALUATION

This section presents functional validation outcomes and analytically derived performance characteristics of the implemented Leave Management System based on architectural design and operational behavior.

6.1 SYSTEM PERFORMANCE CHARACTERISTICS

System performance characteristics are derived from architectural analysis rather than empirical stress benchmarking. Simple readoriented operations such as leave balance retrieval and request listing exhibit low latency due to indexed relational access patterns and minimal business logic processing. In contrast, transactional workflows including leave application submission and approval processing incur higher overhead as they involve multistage validation, entity state transitions, and coordinated database updates within transactional boundaries. The system follows Spring Boot's default embedded server configuration and connection pooling strategy, making it suitable for small to medium organizational workloads under typical usage patterns. Actual throughput and scalability are deployment-dependent and vary based on infrastructure specifications, database performance, and network conditions. Performance benchmarking under concurrent load is identified as future work.

6.2 FUNCTIONAL VALIDATION RESULTS

Functional validation was conducted using scenario-based API testing, covering typical workflows (leave application, approval, retrieval) and error conditions (invalid dates,

overlapping requests, permission violations). Testing focused on validation logic correctness and transactional behavior rather than performance metrics.

Eight core validation rules were verified: date range correctness, prevention of past-dated requests, overlapping leave detection (capturing all intersection cases), leave type validation, employee existence verification, status transition enforcement, and ownership validation. These ensure robust enforcement of business rules and data integrity. Transaction management was tested by simulating faults during approval workflows. Results showed atomicity with automatic rollback on failure, preserving data consistency. Concurrent approvals were handled with proper isolation, allowing only one successful transaction and preventing duplicate processing. Integration tests confirmed coordination between leave and attendance modules, verifying leave approval triggers correct attendance record creation. Although some modules remain partially implemented, the architecture supports modular integration. Compared to manual paper-based processes, the system reduces submission time from minutes to seconds and approval delays from days to immediate managerial response. Automated leave balance tracking prevents arithmetic errors, and overlap detection eliminates scheduling conflicts, highlighting significant operational improvements. Testing was manual and conducted with synthetic data under single-user conditions. Automated test suites, multi-user concurrency validation, and performance benchmarking remain for future work. The system demonstrates functional correctness but is not production ready.

6.3 COMPARATIVE ANALYSIS WITH MANUAL SYSTEMS

Compared to conventional manual leave management processes, the implemented system significantly reduces operational overhead by automating request submission, approval workflows, and balance calculations. Manual processes requiring physical form handling and sequential approvals are replaced by centralized digital workflows with immediate system-level validation. Automated balance tracking eliminates arithmetic errors and synchronization delays inherent in manual systems, while centralized database storage with ACID transaction guarantees enables consistent audit trails and reliable historical access. These capabilities directly address inefficiencies and data consistency issues documented in prior studies of traditional leave management practices.

7. DISCUSSION

7.1 DESIGN VALIDATION AND KEY OBSERVATIONS

The implementation follows a three-tier architecture with Controller Service-Repository layering, exemplifying separation of concerns and maintainability. Declarative transaction management (@Transactional) cleanly handles atomic operations without cluttering business logic. Complexities included formulating accurate overlapping leave detection queries and ensuring transaction isolation to prevent concurrency issues. The DTO pattern decouples API contracts from persistence models, improving flexibility despite initial complexity. Key limitations include the absence of Spring Security integration, plain-text password storage, incomplete leave balance management, and unimplemented weekend/holiday leave calculations. Attendance and payroll modules provide basic structures without full functionality. Testing relies mainly on manual API calls rather than automated suites. This system offers practical exposure to enterprise Java development, RESTful API design, validation frameworks, and transactional consistency, suitable as an academic prototype.

7.2 LIMITATIONS AND CONSTRAINTS

Despite functional correctness, the system is not production-ready. Authentication and authorization mechanisms are absent, with no Spring Security integration, encrypted credential storage, or tokenbased session management. User identity relies on client-provided identifiers, introducing security risks. Notification mechanisms for leave status updates are also unimplemented. The system supports only single-organization deployment and lacks multi-tenant isolation. Leave calculations do not account for weekends, holidays, or partial-day requests, and approval workflows are limited to a single managerial level. Performance optimizations such as caching, pagination, and query tuning remain unaddressed.

7.3 ALIGNMENT WITH LITERATURE FINDINGS

The implementation addresses key shortcomings identified in existing literature, particularly inadequate transaction handling and weak validation frameworks. The use of declarative transactions and multi-layer validation ensures data consistency and prevents invalid submissions at the point of entry. Deferred balance deduction aligns with best practices for maintaining accurate leave balances across approval outcomes.

8. CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

This paper presents a Leave Management System prototype demonstrating layered architecture, comprehensive validation, transaction management, and RESTful API design. The system confirms key software engineering principles through working code. The layered architectural approach supports maintainability and clear separation of concerns, while transaction-coordinated workflows prevent data inconsistencies during concurrent operations or failure scenarios. The system demonstrates the effectiveness of modern enterprise Java frameworks in developing structured and reliable leave management solutions.

8.2 FUTURE WORK

Future work involves integrating Spring Security and BCrypt password hashing, expanding automated test coverage, completing leave balance and calendar-aware leave calculations, enabling multilevel approvals, and extending attendance and payroll functionality toward production readiness. Additional planned enhancements include automated email notifications, multi-tenant deployment support, performance optimizations such as caching and pagination, and deeper integration with payroll and holiday calendar systems.

REFERENCES

[1] A. Adamu, "Employee Leave Management System", *FUDMA Journal of Sciences*, Vol. 4, No. 2, pp. 86-91, 2020.

[2] N. Harshika, P.U. Vardhan, C. Vaishak, A. Akhil, B. Varshitha and S.S. Raoof, "A Multi-Faceted Leave Management Ecosystem Employing AI-Driven Semantic Categorization and Probabilistic Algorithms with Dynamic Schedule Reallocation", *International Journal of Progressive Research in Engineering Management and Science*, Vol. 5, No. 4, pp. 1145-1153, 2025.

[3] S.M. Alade, S. Adejumo and T.J. Alade, "Design and Implementation of a Web based Leave Management System", *International Journal of Computer Applications Technology and Research*, Vol. 11, No. 4, pp. 123-144, 2022.

[4] R. Srinithi and P. Sakthi Murugan, "Employee Leave Management System", *International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering*, Vol. 13, No. 4, pp. 198-202, 2025.

[5] Rashmi, S.S. Dhulugade, P.N. Gaikwad and D.M. Rathod, "Leave Management in Power Apps", *International Journal of Innovative Research in Technology*, Vol. 11, No. 1, pp. 674-679, 2024.

[6] R.S. Birje, R. Benne and A. Unki, "Design and Development of E-Leave Management System", *International Journal of Research Publication and Reviews*, Vol. 6, No. 10, pp. 6464-6472, 2025.

[7] N. Choudhary, A. Khalfe, Y. Khan and M. Ansari, "Leave Management System for AIKTC", *International Research Journal of Engineering and Technology*, Vol. 7, No. 3, pp. 1715-1717, 2020.

[8] M. Singh, P. Singh, R. Singh, S. Singh and S. Gupta, "Leave and Payroll Management System", *Proceedings of International Conference on Computing and Virtualization*, Vol. 8, pp. 62-66, 2017.

[9] R. Sapona, A.H. Thohari and Nelmiawati, "Web-based Leave Management System for Politeknik Negeri Batam", *Journal of Computer Sciences and Engineering*, Vol. 6, No. 4, pp. 72-74, 2020.

[10] Y. Mantri, D.A.R. Kumar and S.U. Kumar, "Emergency Leave Management System with Company Data Analysis", *International Journal of Research in Engineering and Science*, Vol. 11, No. 21, pp. 163-174, 2023.

[11] I. Pathan, B. Nayak, B.S. Nayak, V.B. Dhatrak and K. Daivat, "An Design of AI based Leave Scheduling and Managing Application", *International Journal of Computer Sciences and Engineering*, Vol. 8, No. 4, pp. 97-99, 2020.

[12] S.D. Jadhav, A.A. Ranaware and P.D. More, "Design Steps of Online Leave Management Application System for Academic Institution", *Proceedings of National Conference on Emerging Trends in Science and Advances in Engineering*, Vol. 3, No. 1, pp. 1-6, 2023.

[13] V.K. Kaushik, A.K. Gupta, A. Kumar and A. Prasad, "Student Leave Management System", *International Journal of Advance Research and Innovative Ideas in Education*, Vol. 3, No. 5, pp. 124-131, 2017.