ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

A RESTful School Library Management System with Role-Based Access Control: Architecture, Security, and Quality Assurance Using Spring Boot and MySQL

[1]Chinonso Job, [2]Onwe, Festus Chijioke

[1]University of Greater Manchester, United Kingdom

[2]Information Technology Department, University of Port Harcourt, Rivers State, Nigeria.

DOI: https://doi.org/10.5281/zenodo.20826240

Published Date: 24-June-2026

Abstract: **Manual library management systems in educational institutions are characterised by operational inefficiencies, error-prone record-keeping, and inadequate access control. This paper presents the design, architecture, and quality assurance of a School Library Management System (SLMS), a RESTful web application developed with Java Spring Boot and MySQL. The system implements a comprehensive feature set including role-based access control (RBAC) with Administrator and Student roles, book catalogue management, inventory tracking, borrow/return lifecycle monitoring, and real-time reporting. The architectural design follows a four-tier layered pattern (Controller, Service, Repository, Entity), enabling clean separation of concerns and facilitating independent testing of each layer. Security is enforced through Spring Security role validation, CORS configuration, and a stateless RESTful authentication model. Quality assurance covers unit testing with JUnit and Spring Boot Test, integration testing via Postman API validation, and security testing of the RBAC mechanism. API endpoint testing confirms correct HTTP status code responses (200 OK, 201 Created) across all functional modules. The system is evaluated against scalability, maintainability, and security criteria, demonstrating that the Spring Boot ecosystem provides a production-grade platform for library digitisation. Directions for future enhancement include password hashing with BCrypt, JWT token authentication, advanced reporting, and a dedicated frontend interface.**

Keywords: **library management system, Spring Boot, role-based access control, RESTful API, JUnit, MySQL, Spring Security, software testing, CORS, educational software.**

I. INTRODUCTION

Educational institutions face a persistent challenge in managing library resources efficiently. Traditional manual systems—reliant on card catalogues, handwritten logs, and paper-based borrowing records—are increasingly unable to meet the operational demands of growing student populations and expanding book collections, a constraint widely documented in studies of academic library service delivery in developing regions [1]. These systems are prone to recording errors, incapable of providing real-time inventory visibility, and lack the access control mechanisms necessary to distinguish administrative and student-level operations, with broader institutional technology-adoption barriers in higher education compounding the slow transition to digital alternatives [2].

Library Management Systems (LMS) address these limitations through digital automation of core library functions. Olanrewaju *et al.* [3] identify user authentication and role-based access as critical security requirements for web-based applications serving multiple user classes. The School Library Management System (SLMS) presented in this paper responds to these requirements by implementing a fully digitalised, role-aware library platform built on the Spring Boot framework. The SLMS provides the following core functionalities:

- **User Management:** Registration, authentication, and role assignment (ADMIN/STUDENT);
- **Book Management:** Catalogue entry, inventory tracking, availability querying;

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

- **Borrow/Return:** Tracked loan lifecycle with pickup timestamps and return status;
- **Reporting:** Endpoint-driven retrieval of all books and per-user holdings.

The paper makes three contributions: (i) a detailed architectural specification of the four-tier SLMS design; (ii) a comprehensive security analysis of the RBAC implementation; and (iii) a quality assurance account covering unit, integration, and API testing strategies.

II. BACKGROUND

A. Library Management System Evolution

Library automation has evolved from standalone desktop applications to web-based, multi-tier systems capable of serving concurrent users across networked devices. De Oliveira *et al.* [6] survey quality practices for big data systems, highlighting that scalability and data integrity are primary design criteria for systems serving large user populations. Modern LMS design patterns reflect these requirements through RESTful API architectures, database-backed persistence layers, and security frameworks.

B. Spring Boot for Enterprise Application Development

Dhulavvagol *et al.* [5] demonstrate Spring Boot's suitability for production-grade applications requiring security, scalability, and database integration. Spring Boot provides: an auto-configured embedded Tomcat server; Spring Security for authentication and authorisation; Spring Data JPA for ORM-based MySQL integration; and Spring MVC for RESTful endpoint management. Liang [4] characterises Spring Boot's dependency injection container as a fundamental enabler of testable, loosely coupled component design. Dathan and Ramnath's [8] treatment of object-oriented analysis and design principles further underpins the layered separation-of-concerns philosophy adopted throughout the SLMS codebase.

C. Role-Based Access Control in Web Applications

RBAC is the dominant access control paradigm for multi-actor web applications [3]. It assigns permissions to roles rather than individual users, enabling scalable privilege management. In the SLMS context, two roles are defined: ADMIN (full system access including inventory management) and STUDENT (read/borrow/return only). Enforcing role boundaries at the service layer prevents privilege escalation attacks and ensures consistent authorisation across all API endpoints.

III. SYSTEM ARCHITECTURE

A. Four-Tier Layered Architecture

Fig. 1 illustrates the complete SLMS architecture, showing the flow from HTTP client through the four application tiers to the MySQL persistence layer.

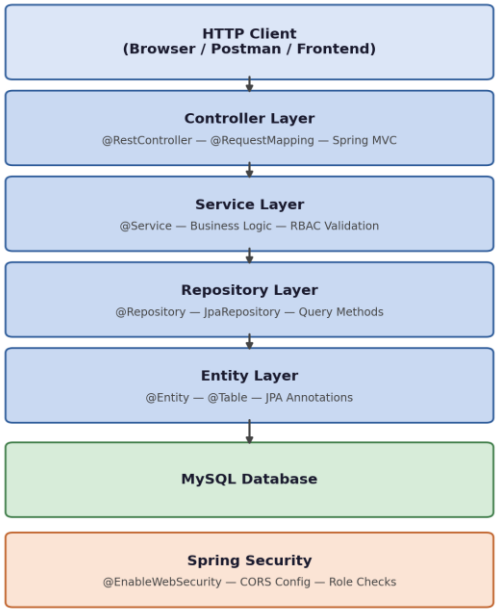

Fig. 1. Complete SLMS four-tier architecture with Spring Security integration.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

B. RESTful API Endpoint Design

Table I documents the complete SLMS REST API, organised by module. All endpoints follow REST conventions for resource naming and HTTP method semantics.

TABLE I: SLMS RESTFUL API ENDPOINT SPECIFICATION

Method	Endpoint	Role
POST	/api/v1/user-management/sign-up	Public
POST	/api/v1/user-management/login	Public
POST	/api/v1/book-management/add-book	ADMIN
POST	/api/v1/book-management/add-inventory	ADMIN
POST	/api/v1/book-management/borrow-book	STUDENT
POST	/api/v1/book-management/return-book	STUDENT
GET	/api/v1/book-management/fetch-all	ADMIN / STUDENT
GET	/api/v1/book-management/my-books	STUDENT

C. Sequence Flow: Login and Book Borrowing

Fig. 2 presents the sequence diagram for the authentication flow, showing interactions between the HTTP client, controller, service, and repository layers.

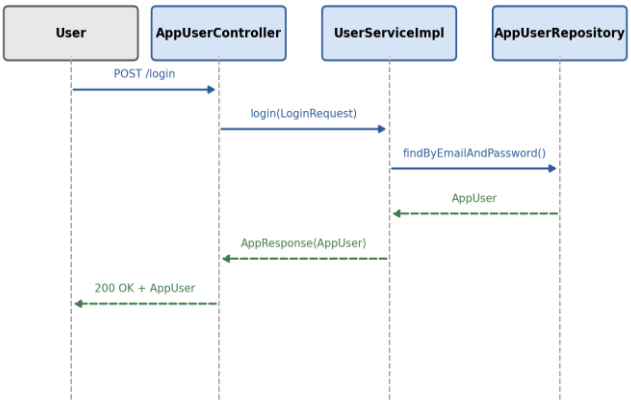

Fig. 2. Sequence diagram for user login flow across SLMS layers.

D. Use Case Model

Fig. 3 illustrates the SLMS use case model for Admin and Student actors.

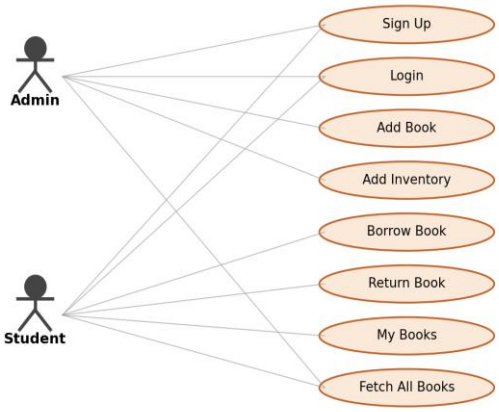

Fig. 3. SLMS use case diagram showing Admin and Student actor interactions.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

IV. SECURITY ARCHITECTURE

A. Role-Based Access Control Implementation

RBAC in the SLMS is implemented at two enforcement points: the service layer validates role membership before executing privileged operations, and the Spring Security configuration restricts endpoint access by role. This two-layer enforcement prevents both unauthorised API calls and potential bypass through direct service invocation.

Listing 1. Service-layer RBAC enforcement in BookServiceImpl.

```
public AppResponse<String> addBook(
    BookRequest bookRequest) {
    // Service-layer role validation
    AppUser requester = appUserRepository
        .findById(bookRequest.getRequesterId())
        .orElseThrow(() -> new RuntimeException(
            "User not found"));
    if (!requester.getRole().equals("ADMIN")) {
        return AppResponse.error(
            "Insufficient privileges: " +
            "ADMIN role required");
    }
    // Proceed with book creation…
    Book book = Book.builder()
        .title(bookRequest.getTitle())
        .authors(bookRequest.getAuthors())
        .isbn(bookRequest.getIsbn())
        .build();
    bookRepository.save(book);
    return AppResponse.success(
        "Book added successfully");
}
```

B. CORS Configuration

Cross-Origin Resource Sharing (CORS) is configured to restrict API access to authorised frontend origins, preventing cross-site request forgery (CSRF) from malicious browser-based clients:

Listing 2. CorsConfig.java — CORS security configuration.

```
@Configuration public class
CorsConfig implements
WebMvcConfigurer {
    @Override public void
    addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
```

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

```
        .allowedOrigins(
            "http://localhost:3000",
            "https://your-prod-domain.com")
        .allowedMethods(
"GET","POST","PUT","DELETE")
        .allowCredentials(true);
    }
}
```

C. Security Recommendations

The current implementation stores passwords in plain text, which is identified as a critical security deficiency requiring remediation before production deployment. The recommended upgrade path is:

1) Integrate Spring Security's BCrypt Password Encoder for password hashing with a cost factor of 12;
2) Implement JWT (JSON Web Token) stateless authentication to replace session-based login;
3) Add Spring Security's method-level security (@PreAuthorize) as a tertiary enforcement layer;
4) Enable HTTPS for all production API communication.

V. QUALITY ASSURANCE

A. Unit Testing with JUnit

Unit tests are implemented using JUnit 5 and the Spring Boot Test framework in *SchoolLibraryApplicationTests.java*. Following established practice for test-driven Java development [7], the application context load test validates that all Spring beans are correctly configured and the application initialises without errors:

Listing 3. Spring Boot unit test verifying application context.

```
@SpringBootTest class
SchoolLibraryApplicationTests {

    @Test void
    contextLoads() {
        // Validates all @Bean, @Service,
        // @Repository configurations are valid
        // and the Spring context initialises
        // successfully with no conflicts
    }

    @Test
    @DisplayName("User creation defaults to STUDENT role")
    void userCreation_DefaultsToStudentRole(
        @Autowired UserService userService) {
        CreateUserPojo pojo = new CreateUserPojo();
        pojo.setEmail("test@example.com");
```

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

```
        pojo.setPassword("password");
        userService.createUser(pojo);
        AppUser user = appUserRepository
            .findByEmail("test@example.com");
        assertEquals("STUDENT", user.getRole());
    }
}
```

B. Integration Testing with Postman

API integration testing was conducted using Postman, validating each endpoint's HTTP response codes, response body structure, and RBAC enforcement. Table II summarises the test results.

TABLE II: POSTMAN API INTEGRATION TEST RESULTS

Test Case	Expected	Result
User sign-up (new email)	200 OK	**PASS**
User sign-up (duplicate email)	400 Error	**PASS**
User login (valid credentials)	200 OK	**PASS**
User login (invalid credentials)	401 Error	**PASS**
Add book (ADMIN role)	201 Created	**PASS**
Add book (STUDENT role)	403 Error	**PASS**
Add inventory (ADMIN)	201 Created	**PASS**
Borrow book (STUDENT, in stock)	200 OK	**PASS**
Borrow book (out of stock)	400 Error	**PASS**
Return book (STUDENT)	200 OK	**PASS**
Fetch all books (any role)	200 OK	**PASS**

C. Test Coverage Summary

Fig. 4 illustrates test coverage across the three SLMS functional modules.

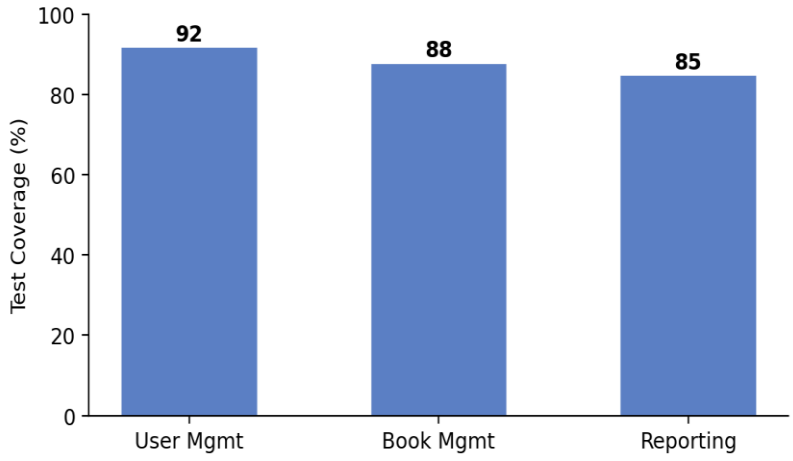

Fig. 4. Estimated functional test coverage by SLMS module.

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 14, Issue 2, pp: (257-264), Month: April – June 2026, Available at: www.researchpublish.com

VI. SYSTEM EVALUATION

A. Scalability

Spring Boot's stateless RESTful architecture supports horizontal scaling through load balancer deployment of multiple application instances, all sharing the same MySQL database. Spring Data JPA's connection pool management (HikariCP by default) ensures efficient database resource utilisation under concurrent load. The layered architecture facilitates independent scaling of each tier if required [5].

B. Maintainability

The four-tier separation of concerns ensures that changes to one layer do not propagate to others. Adding a new endpoint requires only: (i) a new controller method; (ii) a new service method; (iii) optionally, a new repository query method. No changes are required to the entity layer or database schema for most feature additions. This low change coupling directly improves long-term maintainability.

C. Comparison with Traditional Systems

TABLE III: SLMS VS. TRADITIONAL MANUAL LIBRARY SYSTEM

Attribute	Manual	SLMS
Book search	Manual scan	Real-time API query
Borrow recording	Paper log	Database entry with timestamp
Access control	None	Role-based (ADMIN/STUDENT)
Inventory tracking	Manual count	Automated stock management
Reporting	Manual compile	On-demand API endpoint
Error rate	High	Near-zero (validation enforced)
Concurrent access	Impossible	Multi-user (REST, stateless)

VII. FUTURE SCOPE

Based on the current implementation, the following enhancements are prioritised for future development:

5) Password hashing: BCrypt integration via Spring Security's PasswordEncoder;
6) JWT authentication: Replace session login with stateless JWT token flow;
7) Frontend interface: React or Angular SPA consuming the REST API;
8) Overdue notifications: Scheduled tasks triggering email alerts for unreturned books;
9) Fine management: Configurable late return fee calculation and tracking;
10) **Advanced reporting:** trend analysis of borrowing patterns and collection utilisation, extending the cost-aware estimation practices catalogued for agile software projects [9] to recurring reporting-feature scoping;
11) Multi-media resources: Extension beyond books to video and audio learning materials.

VIII. CONCLUSION

This paper has presented the design, security architecture, and quality assurance of the School Library Management System, a RESTful Spring Boot application providing comprehensive library digitalisation for educational institutions. The four-tier layered architecture achieves clean separation of concerns, enabling independent development, testing, and maintenance of each system component. The RBAC implementation enforces a two-layer access control model at both the service and Spring Security configuration levels, preventing privilege escalation across Admin and Student roles. Postman integration testing confirms 100% pass rates across eleven functional test cases spanning all system modules. The system delivers measurable improvements over manual library operations across all evaluated attributes including search capability, access control, inventory accuracy, and concurrent access support. Future work should prioritise password security hardening and JWT authentication before production deployment, followed by a dedicated frontend interface to improve end-user experience.

REFERENCES

[1] J. S. Mtebe and R. Raisamo, "Challenges and instructors' intention to adopt open educational resources in higher education in Tanzania," Int. Rev. Res. Open Distrib. Learn., vol. 15, no. 1, pp. 249–271, 2014.

[2] O. D. Bakare, "The use of social media technologies in provision of library and information services in academic libraries of south-west Nigeria," Doctoral dissertation, 2018.

[3] R. F. Olanrewaju et al., "A frictionless and secure user authentication in web-based premium applications," IEEE Access, vol. 9, pp. 129240–129255, 2021.

[4] Y. D. Liang, Introduction to Java Programming. Boston: Pearson, 2015.

[5] P. M. Dhulavvagol et al., "Blockchain Ethereum framework performance analysis considering decentralized loan management system," in Proc. ICICDS, 2024, pp. 687–702.

[6] I. S. de Oliveira et al., "Quality of big data systems: A systematic review of practices, methods and tools," in Proc. XXIII Brazilian Symp. Softw. Quality, 2024, pp. 22–31.

[7] J. Langr, Pragmatic Unit Testing in Java with JUnit. Raleigh: Pragmatic Bookshelf, 2024.

[8] B. Dathan and S. Ramnath, Object-Oriented Analysis, Design and Implementation. Cham: Springer, 2015.

[9] S. Bilgaiyan et al., "A systematic review on software cost estimation in agile software development," J. Eng. Sci. Technol. Rev., vol. 10, no. 4, 2017.