WRSPM Analysis — Library Management System (LMS)

*Prepared for Capstone Team Submission • September 13, 2025*

**Team**

**Team name: Team 3**

Members (GitHub usernames in parentheses):  
• Kelley, James > https://github.com/kellje  
• Varghese, Shane > https://github.com/shanev05  
• Zelaya, Jonathan > https://github.com/jonpaul007  
• Islam, Rasel> https://github.com/raselislam29

**Overview of the Problem/Project**

Many colleges and community libraries still struggle with fragmented processes for cataloging, circulation, reservations, and fines. The Library Management System (LMS) aims to streamline library operations for librarians and provide a modern, self-service experience for patrons. Core features include catalog search and discovery, user account management, item checkout/return, digital holds and waitlists, fine calculation/payments, and analytics dashboards for resource utilization.

**WRSPM Analysis**

**W — World Assumptions**

Users (librarians and patrons) have reliable internet access and a modern web browser.

Library assets are barcoded or can be labeled with unique IDs (e.g., QR/RFID optional).

The institution has an identity directory (or we will manage accounts within the system).

Data privacy and FERPA-like constraints require role-based access and audit logging.

Payment integration (if used) will rely on a PCI-compliant third-party processor.

**R — User Requirements**

**For Patrons**

Create/manage profile; view current checkouts, holds, and fines.

Search/browse catalog by title, author, ISBN, subject, tags; filter/sort results.

Place holds and joins waitlists; receive notifications (email/SMS) when ready.

Renew eligible items; see due dates and fine accrual in real time.

Access e-resources/links (if applicable) via proxy or embedded viewers.

**For Librarians/Staff**

Circulation desk module for checkout, return, and renewals with barcode/ID input.

Inventory management: add, edit, withdraw items; batch import via MARC/CSV.

Hold queue management and item reservation workflows.

Fine rules configuration (grace periods, rates) and adjustments/waivers with audit.

Reporting/analytics dashboards: circulation stats, popular titles, overdue trends.

**For Admins/IT**

Role-based access control (RBAC) for patrons, staff, and admins.

System configuration (loan periods, item types, branches, holidays/closures).

Backups, restore, and data export; SSO optional (OAuth 2.0/OpenID Connect).

Audit logs for sensitive actions; basic SIEM export capability.

**S — Specifications and Interface Needs**

Responsive web application: patron portal, staff console, and admin console.

RESTful/GraphQL API for core entities (Users, Items, Copies, Loans, Holds, Fines).

Search service with full-text and faceted filters (e.g., Elasticsearch/OpenSearch).

Barcode/QR scanning via webcam or USB scanners; optional RFID integration later.

Notifications via SMTP (email) and pluggable SMS gateway (e.g., Twilio).

Payment integration abstraction (e.g., Stripe) for online fine payments.

Accessibility: WCAG 2.1 AA guidelines; keyboard navigation and ARIA labels.

Internationalization (i18n) and time zone awareness for due dates and notices.

Interfaces to import/export catalog records (MARC21/CSV).

**P — Program (Software Architecture)**

Backend: Java 17 (or latest LTS), Spring Boot, JPA/Hibernate; layered or hexagonal architecture.

Database: PostgreSQL with schema for catalog, circulation, users, and audits.

Search: OpenSearch/Elasticsearch for catalog indexing and discovery.

API security: OAuth 2.0/OpenID Connect (Keycloak/Okta) and RBAC.

Frontend: React or Thymeleaf-based UI (if pure Java, consider Vaadin) with Axios/Fetch.

Testing: JUnit 5, Testcontainers, Mockito; Postman collections for API.

CI/CD: GitHub Actions for build, test, and container publishing.

Containerization: Dockerfile; optional docker-compose for local stack.

M — Hardware (Deployment/Environments)

Developer machines: IntelliJ IDEA (Java 17 SDK), Docker Desktop, Git CLI.

Staging/Prod: Cloud VM or container platform (e.g., AWS EC2/ECS, Azure, or on-prem).

Minimum server sizing (initial): 2 vCPU, 4–8 GB RAM app node; 2 vCPU, 4–8 GB RAM DB.

Storage: Persistent volumes for PostgreSQL; S3/Azure Blob for backups and exports.

Networking: HTTPS via reverse proxy (Nginx/Traefik); domain and TLS certs (Let's Encrypt).

Note: Include a component diagram (API, DB, Search, Auth, Notification) in your repo README.

Key Risks & Mitigations

Risk: Scope creep — Mitigation: Define MVP: catalog search, checkout/return, holds, basic fines; backlog nice-to-haves.

Risk: Data migration complexity — Mitigation: Start with clean seed data; provide CSV/MARC import tool; validate early.

Risk: Auth/SSO delays — Mitigation: Start with local auth; design for pluggable SSO to add later without refactor.

Risk: Performance of search — Mitigation: Index tuning, pagination; load test with realistic datasets early.

Risk: Accessibility gaps — Mitigation: Adopt WCAG checklist and test with screen readers during sprint reviews.

Milestones (Suggested)

Week 1–2: Project setup, domain model, DB schema, basic CRUD APIs.

Week 3–4: Catalog search/indexing, patron portal (browse/search/item details).

Week 5–6: Circulation workflows (checkout/return/renew), holds & notifications.

Week 7–8: Fines rules & payments, staff console, admin settings.

Week 9+: Reports/analytics, accessibility polish, testing, deployment hardening.