Pustakalay

Library Management System

A Project Report Submitted for

VITyarthi - Build Your Own Project

Submitted by:

Rishita Rashi

24BOE10020

Date: November 24, 2025

1. Introduction

In the digital age, the automation of traditional processes is essential for efficiency and accuracy. "Pustakalay" is a comprehensive Library Management System designed to streamline the operations of a library. It bridges the gap between physical resources and digital management by providing a platform for students, faculty, and administrators to interact with the library's inventory seamlessly.

This project leverages Java for robust backend processing and React for a modern, responsive user interface, demonstrating a full-stack approach to solving real-world problems.

2. Problem Statement

Traditional manual library management systems suffer from several inefficiencies:

- **Data Redundancy & Errors:** Manual record-keeping is prone to human error and duplicate entries.
- **Time-Consuming:** Searching for books and processing issues/returns manually consumes significant time.
- **Lack of Real-time Status:** Users cannot easily verify if a book is available without visiting the library.
- **Poor Analytics:** It is difficult to track borrowing trends or inventory health with paper-based systems.

Pustakalay addresses these issues by automating data management, providing instant search capabilities, and maintaining real-time transaction records.

3. Functional Requirements

- **User Management:**
  - Registration for new users (Students).
  - Login authentication for Students, Faculty, and Admins.
  - Role-based access control.
- **Book Management:**
  - Admin can Add and Remove books.
  - Users can Search for books by Title or Author.
  - View detailed book information (Genre, Availability).
- **Transaction Management:**
  - Issue a book (updates status to "ISSUED").
  - Return a book (updates status to "AVAILABLE").
  - View personal transaction history.

4. Non-functional Requirements

- **Performance:** The system responds to search queries and transactions in under 200ms.
- **Usability:** The User Interface is intuitive, responsive, and accessible via standard web browsers.
- **Reliability:** The system handles invalid inputs gracefully with proper error messages (e.g., "Book not found").
- **Maintainability:** The code follows a modular structure (Model-View-Controller pattern) to allow easy updates.
- **Security:** Basic authentication ensures only registered users can perform transactions.

5. System Architecture

The system follows a Client-Server architecture:

- **Frontend (Client):** Built with React.js and Tailwind CSS. It handles user interactions and communicates with the backend via REST API.
- **Backend (Server):** Built with Java (JDK 8+) using `com.sun.net.httpserver`. It processes requests, executes business logic, and manages data.
- **Data Layer:** In-memory data structures (Lists/Maps) simulate a database for this project scope.

6. Design Diagrams

Use Case Diagram

usecaseDiagram actor Admin actor Student actor Faculty package Pustakalay { usecase "Login" as UC1 usecase "Search Book" as UC2 usecase "Issue Book" as UC5 usecase "Return Book" as UC6 } Admin --> UC1 Student --> UC1 Student --> UC2 Student --> UC5 Student --> UC6 Faculty --> UC1 Faculty --> UC5

Class Diagram

classDiagram class User { +String userId +String name +String role } class Book { +String bookId +String title +boolean isAvailable } class Transaction { +String transactionId +String userId +String bookId } class LibraryService { +addBook() +searchBooks() } LibraryService "1" -- "*" Book

Sequence Diagram (Issue Book)

sequenceDiagram User->>Frontend: Click "Issue Book" Frontend->>Backend: POST /api/issue Backend->>LibraryService: checkAvailability() LibraryService-->>Backend: true Backend->>TransactionService: createTransaction() Backend-->>Frontend: Success Response Frontend-->>User: Show "Book Issued" Alert

7. Design Decisions & Rationale

- **Java HTTP Server:** Chosen to keep the backend lightweight and dependency-free, avoiding the overhead of heavy frameworks like Spring for this specific scope.
- **React (CDN):** Used to demonstrate modern frontend capabilities without requiring a complex Node.js build pipeline on the host machine.
- **In-Memory Storage:** Selected for simplicity and speed, allowing focus on logic implementation rather than database configuration.

8. Implementation Details

The project is structured into three main packages:

- `com.vityarthi.pustakalay.model`: Contains POJO classes (Book, User, Transaction).
- `com.vityarthi.pustakalay.service`: Contains business logic (LibraryService, UserService).
- `com.vityarthi.pustakalay.api`: Contains REST API handlers (LoginHandler, IssueHandler).

Test Case	Steps	Expected Result	Status
User Registration	Enter valid details -> Click Register	Success Alert	Pass
Book Search	Enter "Malgudi" in search bar	"Malgudi Days" displayed	Pass
Issue Book	Click Issue on available book	Status changes to ISSUED	Pass

9. Results

The Pustakalay Library Management System was successfully implemented and tested. Key results include:

- **User Authentication:** Users can successfully register and log in. The system correctly identifies roles (Student vs. Admin).
- **Book Management:** The dashboard correctly displays the list of books fetched from the backend API.
- **Transaction Processing:** Books can be issued and returned. The availability status updates in real-time across the system.
- **API Performance:** All API endpoints respond within acceptable time limits, ensuring a smooth user experience.

10. Testing Approach

We employed **Manual Black-Box Testing**. Key test cases included:

- Validating user login with correct/incorrect credentials.

- Verifying book availability updates immediately after issuance.
- Ensuring unauthorized users cannot access protected API endpoints.

11. Challenges Faced

- **CORS Issues:** Connecting the React frontend to the Java backend initially caused Cross-Origin Resource Sharing errors, which were resolved by adding proper headers in the Java `APIServer`.
- **State Management:** Keeping the frontend UI in sync with the backend state (e.g., updating the button from "Issue" to "Return" immediately) required careful React state handling.

12. Learnings & Key Takeaways

- Gained deep understanding of how REST APIs work under the hood by implementing one from scratch in Java.
- Learned how to integrate a Java backend with a modern JavaScript frontend.
- Understood the importance of modular design in keeping code maintainable.

13. Future Enhancements

- **Database Integration:** Replace in-memory storage with MySQL or MongoDB for persistent data.
- **Barcode Scanner:** Integrate physical barcode scanning for faster book issuing.
- **Fine Calculation:** Implement logic to calculate fines based on return dates.

14. References

- Java Documentation: https://docs.oracle.com/en/java/
- React Documentation: https://reactjs.org/
- Tailwind CSS: https://tailwindcss.com/