# Introduction

Ineffective inventory tracking, laborious borrowing processes, and poor user management are just a few major operational problems the Library Management System (LMS) created for this project seeks to solve. The project's primary goal is to create an automated, dependable, and secure system to effectively and efficiently manage library operations. The system ensures smooth inventory management and transaction processing by facilitating user-friendly interactions for general users and library administrators.  
  
The chosen solution is significant because it uses modern software engineering techniques to modernise traditional library management systems. The project offers a reliable and scalable solution to address standard library requirements by combining Java Spring Boot for backend development, MongoDB for data storage, and JWT-based authentication for security.

# System Architecture

The LMS employs a layered architecture, ensuring clear separation of concerns and facilitating maintainability and scalability. The key components include:

* Backend
* API Layer
* Database
* Security

A diagram of a software system

AI-generated content may be incorrect.

## Backend

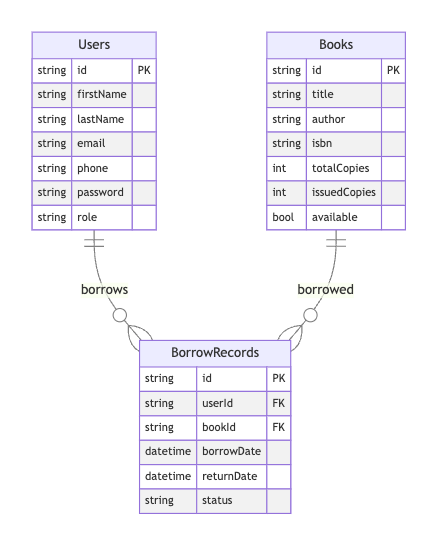
The backend is developed using Java with the Spring Boot framework, which provides rapid application development capabilities, embedded servers, and built-in dependency injection. It encapsulates business logic and ensures secure data manipulation and retrieval.

## API Layer

RESTful APIs are implemented to manage interactions between users and the backend system. APIs handle user authentication, book management, and transaction processing requests, employing appropriate HTTP methods and response codes to ensure clarity and consistency.

## Database

MongoDB, a NoSQL database, is chosen for its flexibility and scalability. The database structure includes collections for users, books, and borrow records. Schematic design promotes efficient data retrieval and integrity, supporting indexing and normalisation techniques.



## Security

The security layer employs JWT-based authentication facilitated by Spring Security. Role-based access control ensures that authorised users can only perform specific actions, enhancing system security and integrity.

# Implementation

## Backend Development

The backend of this Library Management System is built using Java and Spring Boot, a powerful and popular combination within enterprise application development. Spring Boot is chosen because it efficiently bootstraps Spring applications by abstracting away the complex configuration and allowing developers to focus directly on business logic.

In this system, the backend is systematically structured into multiple layers:

* Controller Layer: This layer serves as the first point of interaction, mapping API requests to service methods. For example, the BookController manages requests related to book management and routes these to the appropriate service classes.
  + Controllers abstract HTTP request handling and enforce role-based security through Spring Security annotations (@PreAuthorize).
  + Clearly defined RESTful principles (POST, GET, PATCH, DELETE) ensure predictable and standardised API behaviour, significantly improving maintainability and usability.

A computer screen shot of a program

AI-generated content may be incorrect.

* Service Layer: This encapsulates business logic, keeping the controllers lean and focusing on application-specific functionality. For example, BookServiceImpl implements core operations such as creating, updating, retrieving, and deleting book records.
  + Isolates application logic, simplifying debugging and unit testing.
  + Employs domain-specific validation, such as verifying whether a book can be safely deleted, thus enforcing business integrity rules.

A screenshot of a computer code

AI-generated content may be incorrect.

* Repository Layer: This layer provides a direct interface to the database using Spring Data MongoDB. This abstraction leverages Spring Data's repository pattern to interact effortlessly with MongoDB without verbose queries.
  + Automatically generated queries eliminate boilerplate code, reducing errors and maintenance overhead.
  + Extensible custom queries seamlessly integrate within the repository interface, such as checking if borrowed records exist.

A close-up of a text

AI-generated content may be incorrect.

## Database Interaction

MongoDB stores critical data, including user information, book inventory details, and transaction records. The integration of Spring Data MongoDB simplifies interactions with the database, providing automated query derivation and object-document mapping. Database schema design emphasises efficient querying and retrieval through indexing and normalisation, ensuring rapid responses and optimal performance.

Example entity representation:



* Flexible schema: Allows rapid prototyping and easy adaptability to future changes, essential for iterative agile development processes.
* Scalability: Horizontal scaling potential suits real-world, large-scale deployments.
* Performance: MongoDB indexing and query optimisation techniques enhance data retrieval efficiency.

The integration with Spring Data further streamlines interactions through automatic mapping and query derivation, significantly reducing complexity.

## API Creation

RESTful APIs, developed using Spring Web, facilitate interactions between frontend interfaces and backend logic. Endpoints are clearly defined, adhering to REST principles, and streamline operations such as user registration, authentication, book management, and borrowing transactions. Swagger integration provides clear API documentation, enhancing usability and ease of integration.

Each API response follows a uniform pattern encapsulated within the ResponseEntityDto, ensuring standardised, predictable communication:

A screen shot of a computer program

AI-generated content may be incorrect.

* Consistent API design: Centralizes and standardises response handling, enhancing client integration and reducing ambiguity in response interpretation.
* Error handling and reporting: Coupled with a global exception handler (GlobalExceptionHandler), ensures all exceptions are translated into clear, client-readable responses.

## Security Implementation

Security considerations are paramount in system design. JWT-based authentication and role-based authorisation via Spring Security ensure robust access control:

A screenshot of a computer code

AI-generated content may be incorrect.

* Stateless design: JWT enables stateless authentication, significantly improving scalability and resilience.
* Fine-grained role control: @PreAuthorize annotations enforce method-level security, making it easier to manage permissions precisely where needed.

## Error and Exception Handling

Centralised global error handling via @ControllerAdvice ensures comprehensive and uniform error management:

A screenshot of a computer program

AI-generated content may be incorrect.

* Provides a single point for managing exceptions across the entire application, simplifying debugging and logging.
* Ensures error responses are standardised, greatly enhancing client application usability and fault tolerance.

## DTO and Mapper Layer

Data Transfer Objects (DTOs) effectively decouple internal entities from external API contracts, utilising custom mapper components like BookMapper for data transformation:

A computer screen shot of text

AI-generated content may be incorrect.

* Separation of concerns: Clearly distinguishes data representation from business logic, maintaining clarity and simplifying maintenance.
* Flexibility: Facilitates API versioning and future modifications with minimal disruption to internal logic.

# Challenges and Solutions

## Data Integrity

* Challenge: Maintaining data integrity during critical operations such as deleting books.
* Solution: Implement pre-deletion validation checks to ensure books with active borrow records cannot be deleted and thus maintain data consistency.

## Secure Authentication

* Challenge: Ensuring secure user authentication and authorisation.
* Solution: Adoption of JWT authentication integrated with Spring Security to ensure secure handling of user credentials and strict enforcement of role-based access controls.

## Scalability

* Challenge: Ensuring system performance under high concurrent user loads.
* Solution: Optimization of MongoDB queries and effective indexing strategies to handle increased load and maintain rapid response times.

# Results

The LMS successfully achieves the outlined objectives, providing secure and efficient management of library resources. Testing demonstrated robust functionality:

* User Management: Effective authentication and authorisation procedures validated by multiple test scenarios.

If there isn’t any admin in the system, add admin to the database when starting the application.

A screenshot of a computer

AI-generated content may be incorrect.

Admin Login Request

A screenshot of a computer

AI-generated content may be incorrect.

Users sign up for the library system.

A screenshot of a computer

AI-generated content may be incorrect.

User login to the system

A screenshot of a computer

AI-generated content may be incorrect.

* Book Management: Comprehensive CRUD operations are facilitated through administrative interfaces and verified via successful API responses and database validations.

Add the book to the system; only the admin can add books. Therefore, we need to pass the jwt token, which checks the role and permits the request in the backend.

A screenshot of a computer

AI-generated content may be incorrect.

Update book

A screenshot of a computer

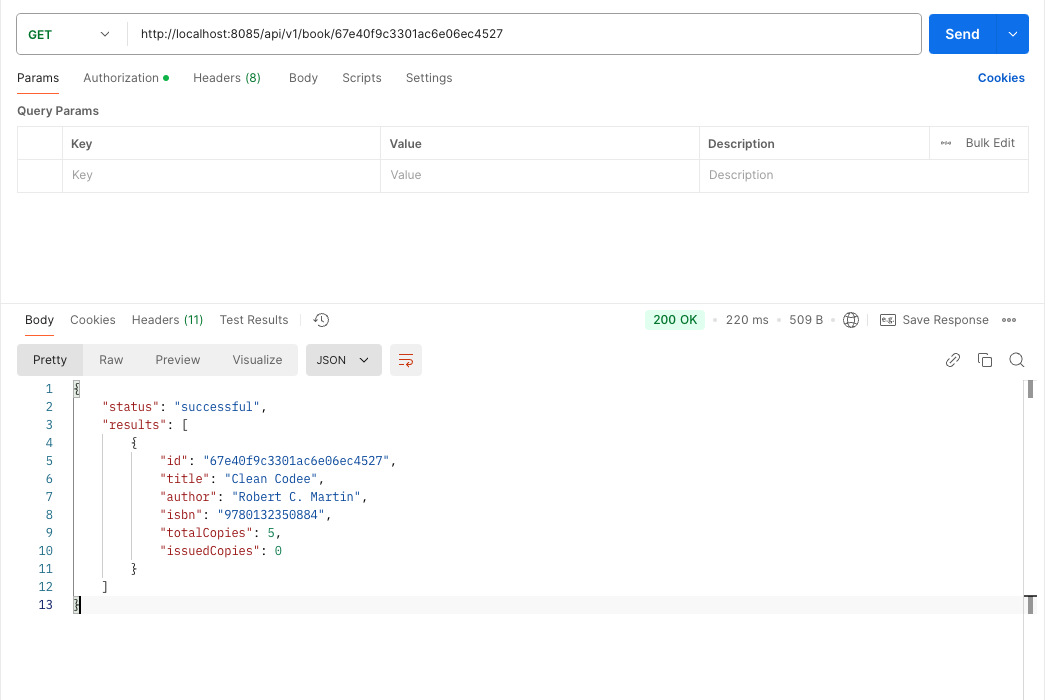
AI-generated content may be incorrect.

Get an invalid book using the book ID.

A screenshot of a computer

AI-generated content may be incorrect.

Get book



Get all books

A screenshot of a computer

AI-generated content may be incorrect.

* Borrowing and Returning: Successfully validated through API calls, ensuring accurate data representation in the borrow records and real-time inventory tracking.

Borrow invalid book

A screenshot of a computer

AI-generated content may be incorrect.

Borrow a book

A screenshot of a computer

AI-generated content may be incorrect.

Handover a book

A screenshot of a computer

AI-generated content may be incorrect.

Screenshots from application interfaces and API responses are provided to demonstrate successful functionality and operational efficiency.

# Conclusion and Future Work

The Library Management System effectively addresses critical library management challenges through robust architectural design and advanced software engineering techniques. The project’s successful implementation showcases significant automation, security, and usability advancements, greatly benefiting libraries seeking modern operational efficiencies.

* Recommendations for future improvements include:
  + Integration of notification services for user updates.
  + Implementation of advanced analytics for administrative insights.
  + Enhanced graphical user interfaces to improve user experience.
  + Real-time inventory management using WebSocket technologies.

These enhancements further improve system usability, scalability, and operational effectiveness.