# SD$^2$: Software Design Document
# TOPIC : LIBRARY MANAGEMENT SYSTEM
# TEAM : 12

## 1. PROJECT OVERVIEW

### INTRODUCTION:

A library is a place where a large number of books are available for patron use. It serves as the institution's brain. It enhances learners' exposure to intellectual and spiritual civilization. The abundance of books and study materials encourages students to broaden their knowledge and perspectives. It teaches learners how to promote their ideas in unique ways. Technological advancements have led to the need to transform traditional libraries into digital ones. The traditional techniques used in libraries are frequently labor and time intensive. Manual record-keeping and physical assistance are required for all actions, such as borrowing and returning books. Routine tasks reduce the library's effectiveness and bore staff members. When a book is borrowed, the librarian must record its information in a notebook, including the ID, title, and author. Data is collected and saved for future use. When the user receives the book, they must also fill out the notebook with their ID, distribution and renewal dates, and book ID.

When a book is returned, the admin updates the books with the user's information and places it back on the shelf. Each book must have an admin-provided tag and ID, and be properly labeled and organized on the shelves. When a book disappears or is stolen, the administration is concerned and must verify the penalties before taking action. Library staff may experience decreased motivation and productivity due to the time and effort required for these tasks. To address these issues, we recommend this system that integrates libraries and technology.

### PROBLEM SOLUTION:

The proposed solution is to develop an online Library Management System that automates various processes and provides a user-friendly interface for both administrators and users. The library management system aims to address the problems faced with traditional manual methods by incorporating the following features:

- Add, update, and delete book records
- Search for books by title, author, genre, and ISBN
- Track book availability and lending status

User Management:

- User registration and authentication
- User profile management
- Track user borrowing history and outstanding fines

Lending and Return Process:

- Online book request and approval system
- Book return and fine.

The project's goal is to create an online library management system that automates and simplifies a number of the procedures required in running a library. Java, the Spring Boot framework, MySQL, and front-end technologies including HTML, CSS, Bootstrap, and jQuery are all used in the construction of the system.

**MAIN USE CASES:**

User Registration and Authentication: Users can sign up, log in, and manage their accounts. Admin users have additional privileges to manage the system.

Book Management: Admin users can add new books, update book details (title, author, genre, ISBN, etc.), and delete books from the system.

Book Issuing and Returning: Users can search for books by title, author, or genre, and issue available books. Admin users can grant issue requests, mark books as issued or returned, and handle book renewals.

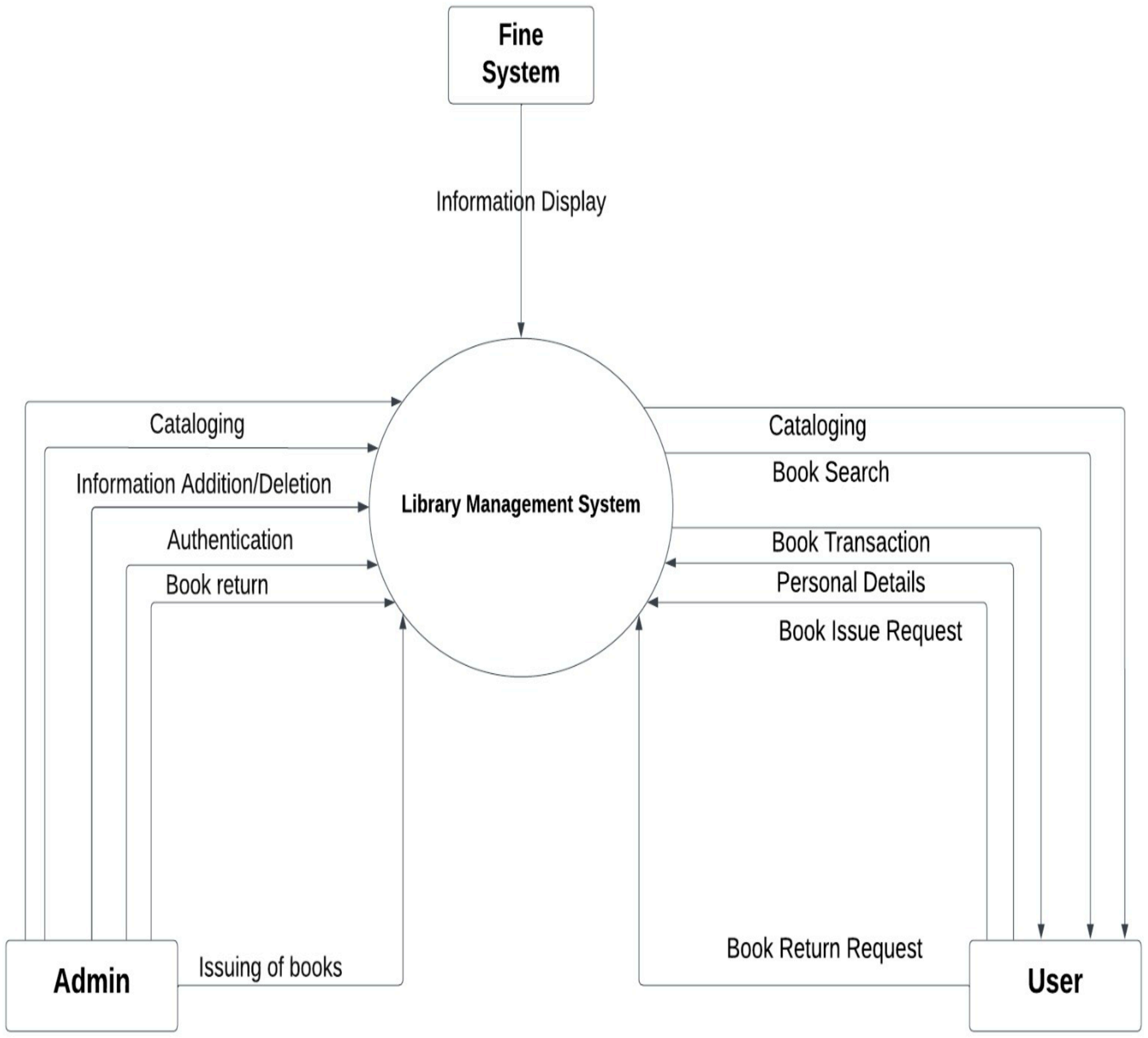
Fines and Reminders: The system automatically calculates and updates fines for overdue books. It also sends email reminders to users when their book due dates are approaching.

Reports and Analytics: The system generates various reports for both admin and users, including available books, book usage statistics, author/genre popularity, user issue history, and fine reports.

User Profile Management: Users can update their profile information, change passwords, and view their issued/returned book history and any outstanding fines.

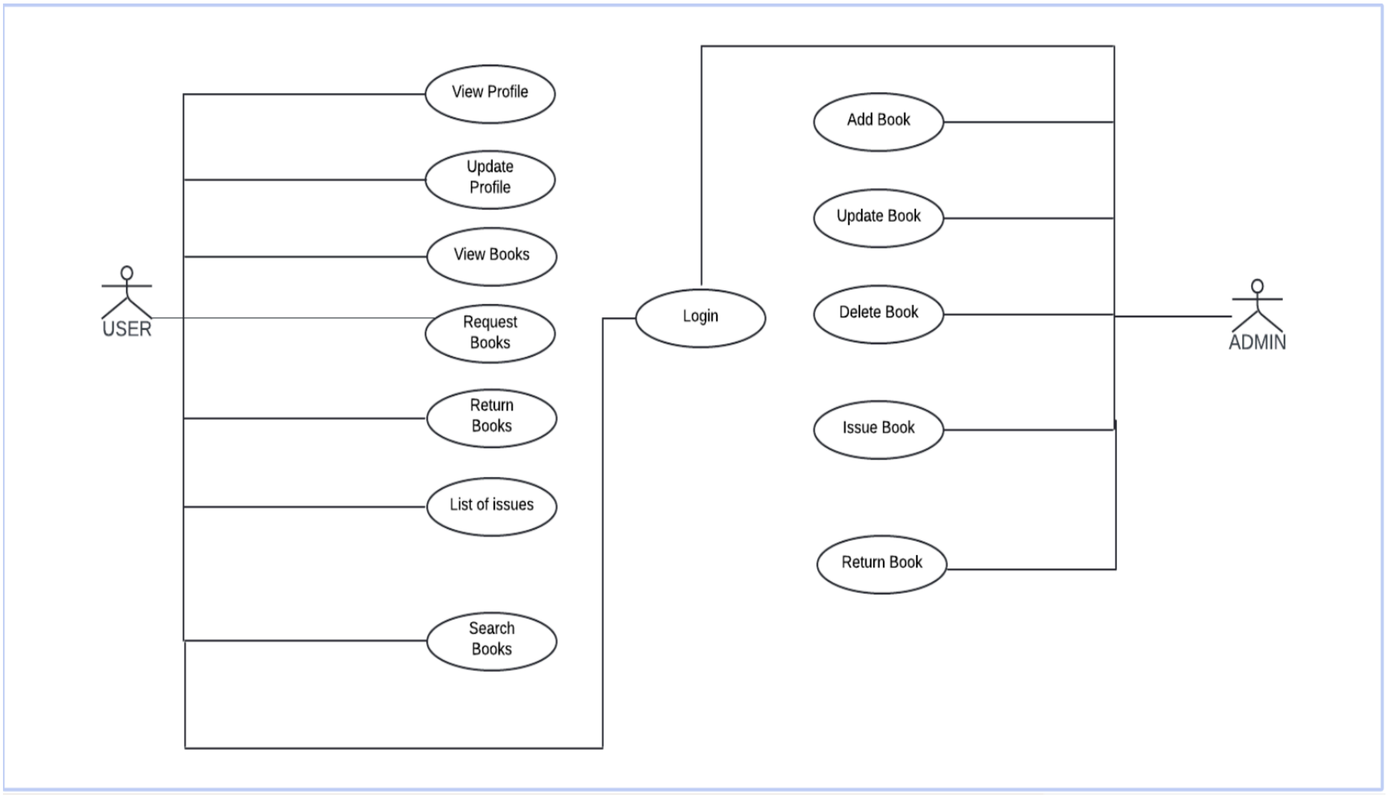
**General Model:**

**Context Diagram:**



The Library Management System will interact with two main actors: Admin and User.

**USE CASE DIAGRAM:**



The use case diagrams will depict the various functionalities available to admin and users, such as Add Book, Update Book, Delete Book, Issue Book, Return Book as Admin side, and Verify Profile and update profile and view Books, Borrow books, Return books etc..

The project aims to provide an efficient and user-friendly platform for managing library operations, improving the overall experience for both library staff and members. It automates various manual processes, reduces human effort, and ensures accurate record-keeping and inventory management.

## 2. Architectural overview

The Library Management System will be designed as a web-based application following a client-server architecture. The system will have the following main components:

**Frontend:**

The frontend will be a responsive web application built using HTML, CSS, JavaScript, and frontend frameworks/libraries like Bootstrap and jQuery.

It will provide a user-friendly interface for both administrators and library users.

Admin interface will include features for book management, user management, and reporting.

User interface will include features for book search, borrowing, returning, and account management.

**Backend:**

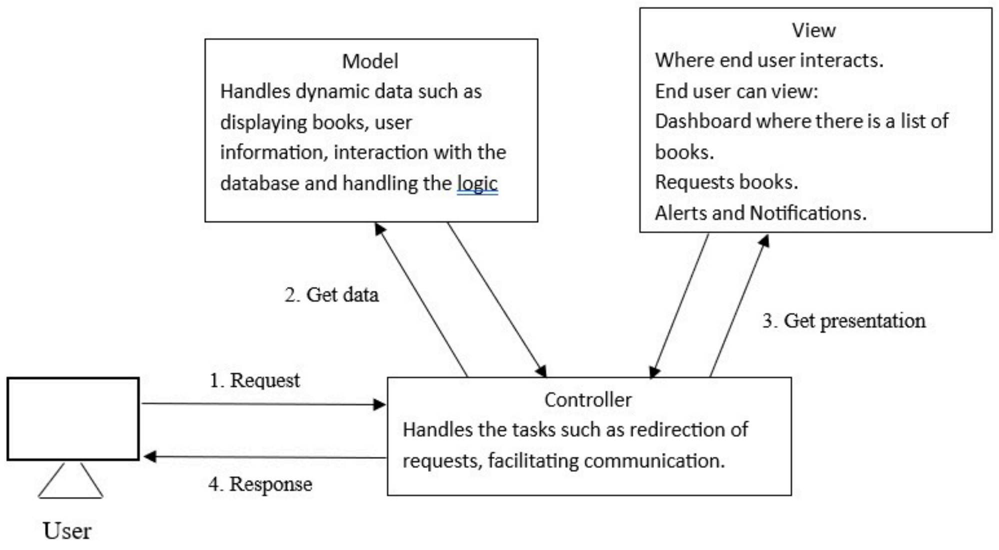
The backend will be developed using the Spring Boot framework in Java.

The backend will implement business logic, data validation, and authentication/authorization mechanisms.

Spring Security will be used for user authentication and authorization.

The system will use a relational database management system, such as MySQL, to store data.

**MVC :**


Model
Handles dynamic data such as displaying books, user information, interaction with the database and handling the logic
View
Where end user interacts.
End user can view:
Dashboard where there is a list of books.
Requests books.
Alerts and Notifications.
2. Get data
3. Get presentation
1. Request
Controller
Handles the tasks such as redirection of requests, facilitating communication.
4. Response
User

The image depicts a high-level architectural overview of a library management system, highlighting its key components and interactions.

**User:** This identifies the system's end-user, such as a library patron or staff member.

**Model:** This component manages the system's dynamic data, such as displaying books, user information, and interacting with the database to retrieve and manipulate data.

**Controller:** This component acts as an intermediary between the user and the model, handling tasks such as request redirection, communication facilitation, and user interaction processing.

**View:** This component represents the end-user interface, which allows the user to browse a list of books, make requests, and receive alerts and notifications.

The sequence of interaction is as follows:

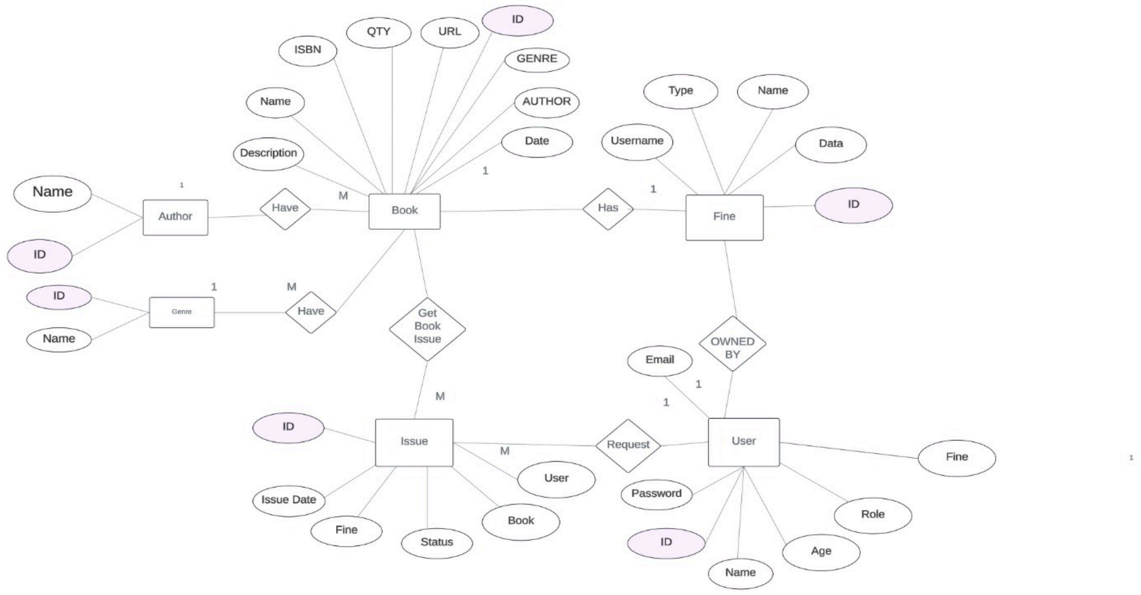
The user submits a request to the system.

The Controller receives data from the Model.

The Controller then renders the necessary presentation to the View, with which the end user can interact.

This architectural design adheres to the Model-View-Controller (MVC) pattern, which is a popular design pattern for developing web applications and software systems. The separation of concerns among the Model, View, and Controller components improves the modularity, maintainability, and scalability of the library management system.

**ER DIAGRAM:**



The entity-relationship (ER) diagram represents the data model for a library management system. the key architectural components and relationships can be summarized as follows:

**Book:**

The central entity in the system is the "Book" entity, which has various attributes such as ID, ISBN, QTY, URL, Genre, Author, Date, Username, and Fine.

The Book entity has relationships with other entities, including "Get Book Issue" and "Fine".

**Author:**

The "Author" entity is associated with the Book entity, where each book has an author.

The Author entity has attributes such as ID, Name, and Description.

**Genre:**

The "Genre" entity is also associated with the Book entity, where each book belongs to a specific genre.

The Genre entity has an ID and Name attribute.

**User:**

The "User" entity represents the users of the library management system.

The User entity has attributes such as ID, Name, Username, Password, and Role.

The User entity has a relationship with the "Request" entity, where users can request books.

The User entity also has a relationship with the "OWNED BY" entity, which associates the user with the books they own.

**Issue:**

The "Issue" entity represents the process of borrowing a book by a user.

The Issue entity has attributes such as ID and Issue Date, and is associated with the Book and User entities.

**Request:**

The "Request" entity represents the process of a user requesting a book.

The Request entity is associated with the User and Book entities.

**Fine:**

The "Fine" entity represents the fines associated with books, which can be applied to users.

The Fine entity is associated with the Book and User entities.

This ER diagram provides a high-level architectural overview of the library management system, highlighting the key entities, their attributes, and the relationships between them. This information can be used to design the database schema and develop the core functionality of the library management application.

**2.1 Subsystem Architecture**

The Library Management System (LMS) has four layers that we will define in order to develop a subsystem dependency diagram: the application-specific, application-generic, middleware, and system-software layers. Packages corresponding to various system components will be present at every tier.

**Application-Specific Layer**: This layer contains modules dedicated to managing specific aspects of the library system. For example, the User Management module handles user-related tasks like registration and authentication, while the Book Management module deals with book-related functionalities such as adding, deleting, and searching for books. Additionally, the Issue Management module is responsible for book borrowing and returning, while the Request Management module manages borrowing requests made by users.

**Application-Generic Layer**: Here, we have packages that provide generic functionalities that can be reused across different applications. For instance, the Authentication package offers mechanisms for user login, logout, and session management. The Database Access package abstracts database operations like creating, reading, updating, and deleting user data, book data, and fine data. Lastly, the Logging package records system events and errors for monitoring and debugging purposes.

**Middleware Layer**: Acting as a bridge between the application-specific and system-software layers, this layer facilitates communication and provides essential services. The Service Layer exposes APIs for the application-specific functionalities to interact with the system-software layer. Meanwhile, the Communication Protocol package handles communication protocols like HTTP or RPC for seamless interaction between the application-specific and system-software layers.

**System-Software Layer**: This layer comprises the underlying infrastructure and system software components essential for the LMS to operate efficiently. The Database Management package oversees database operations such as storage, retrieval, and indexing. On the other hand, the Operating System Interface package provides interfaces for interacting with the underlying operating system, including file system operations and process management.

The architectural styles employed in LMS include:

- **Layered Architecture**: This design promotes modularity, maintainability, and scalability by segregating functionalities into distinct layers.
- **Client-Server Architecture**: By adopting a client-server model, LMS centralizes data management and enhances scalability to accommodate multiple users. Clients interact with the system through a front-end interface, while the server processes requests and manages data on the backend.

These architectural choices are made to ensure the system's flexibility, reliability, and performance while addressing key requirements of the LMS.

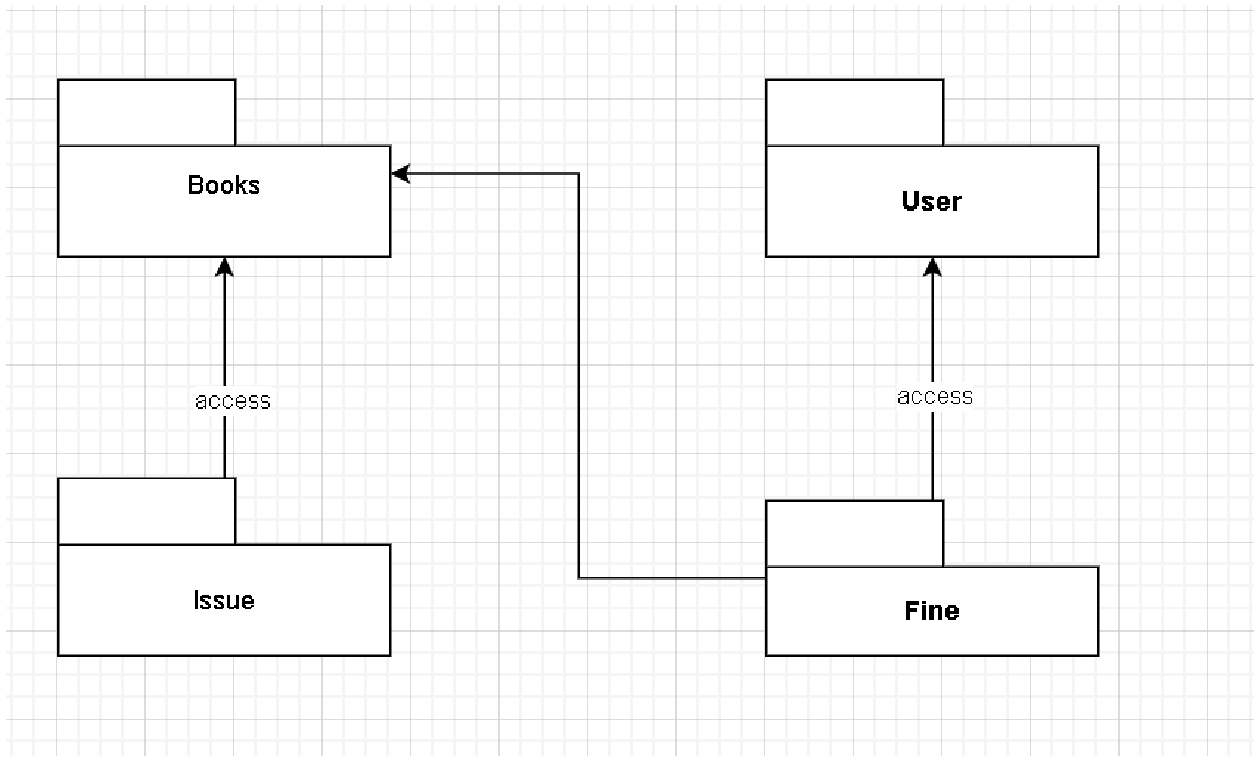
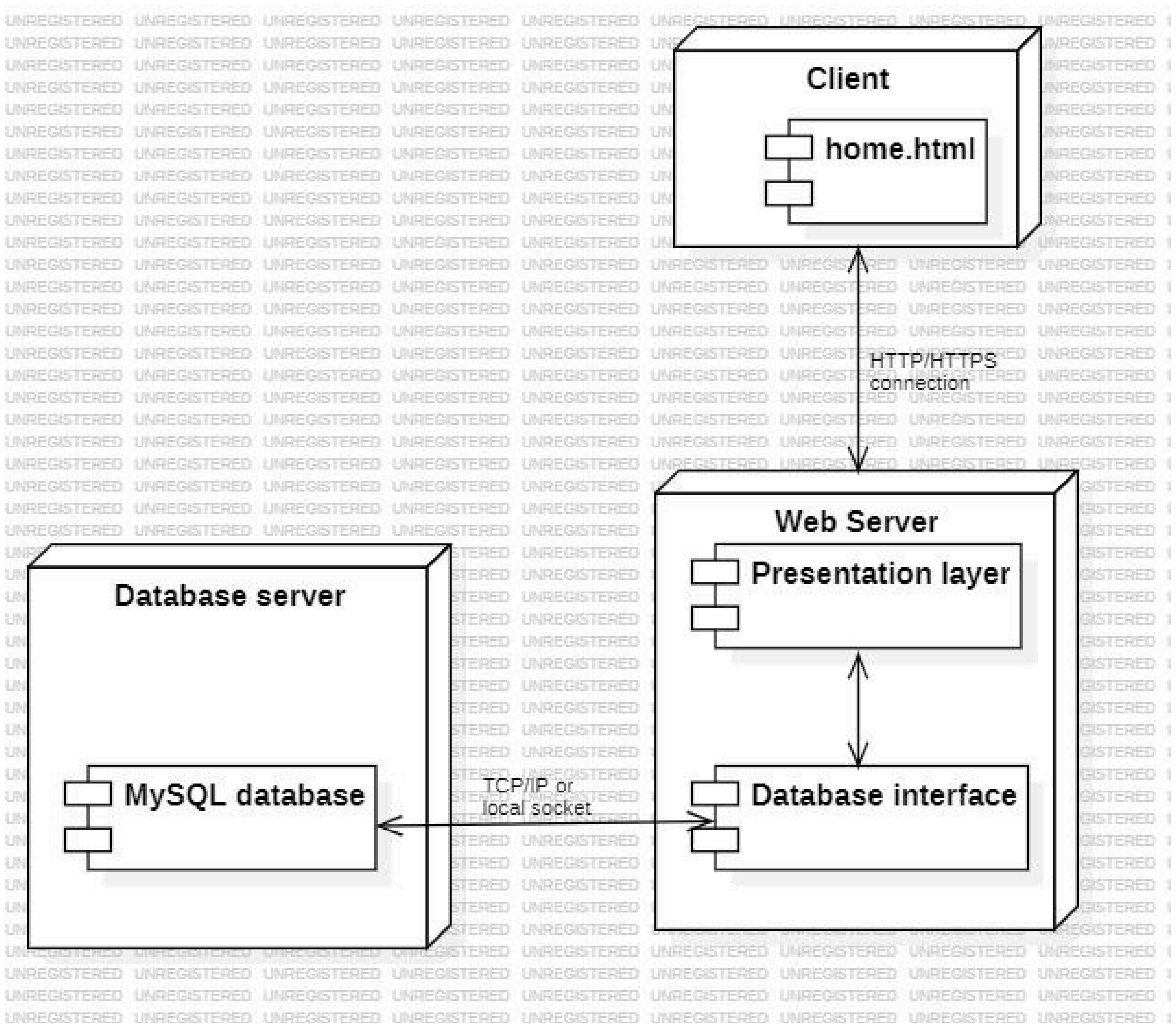


**Fig: UML Package Diagram**

### 2.2 Deployment Architecture



### 2.3 Persistent Data Storage

To handle books, users, transactions, and other relevant information, a library management system must maintain a variety of data. Here's a proposed approach for storing this data using a relational database:

Database schema:

The tables involved in the library management system are as follows:

**Book:** The Book table contains information about the books available in the library and the attributes of the book table are as follows:

book_id (Primary Key): Unique identifier for each book.

date: The date on which this book information is inserted into the database.

description: Describes the book.

isbn: International Standard Book Number for the book.

qty: Number of copies available in the library

title: Title of the book.

author_id (Foreign Key): References the author who wrote the book in the 'Author' table.

file_id (Foreign Key): References the file associated with that book in the 'File' table.

genre_id (Foreign Key): References the genre of the book in the 'Genre' table.

**Author:** The Author table includes the information about the author and the attributes of the author table are as follows:

author_id (Primary Key) : Unique identifier for each author.

author_name: Name of the author.

**Genre:** The Genre table contains information about the genre of the book and the attributes include:

genre_id (Primary Key): Unique identifier for each genre.

genre_name: Name of the genre.

**File:** The File table keeps the information of the image files uploaded for the cover page of the book. The attributes of the file table are as follows:

file_id (Primary Key): Unique identifier for each file.

file_name: Name of the file.

file_type: Type of the file.

data: BLOB data of the image file which is the cover page of the book.

**Issue:** The Issue table keeps track of the borrowing history of books and the attributes are as follows:

issue_id (Primary Key): Unique identifier for each issue.

issue_date: Date on which the book is issued.

fine: Fines if any, accrued for late returns.

status: Status of the book (Requested, Issued, Returned).

book_id (Foreign Key): References the book_id in the book table, indicates the book being borrowed.

user_id (Foreign Key): References the user_id in the User table, indicates the user who borrowed the book.

**User:** The User table contains information about the library members.

The attributes of the user table are as follows:

user_id (Primary Key): Unique identifier for each user.

age: Age of the user.
email: Email address of the user.
fine: Fines if any, accrued for late returns by the user.
is_enabled: Confirms that the user is authorized.
name: Name of the user.
password: Password set by the user.
role: Role of the user. (ROLE_USER, ROLE_ADMIN)
username: Username of the user.
**Token:** The Token table contains tokens associated with the authentication and the attributes are as follows:
token_id (Primary Key): Unique identifier for each authentication.
confirmation_token: Token generated when authentication.
created_date: Date on which the token is created.
user_id (Foreign Key): References the user_id in the User table, who signs up.

## 2.4 Global Control Flow

Procedural or Event-Driven:
An LMS can be both procedural and event-driven, depending on the specific functionality:

- Procedural: Basic administration tasks, such as adding new books to the catalog, updating book information, and processing user registrations, may be assigned to a procedural control flow. This means that these duties are often carried out in a linear order of phases, usually using an administration interface.

- Event-Driven: User interactions with the system, such as looking for books, borrowing or returning them, and getting notifications, are best handled using an event-driven model. In this approach, the system waits for user actions (events) and replies correctly. For example, when a book is returned, an event is generated, which updates the book's availability status and may notify users who have reserved it.

**Time Dependency:**

- Timer-Controlled Actions**:** Certain aspects of an LMS may be time-dependent, such as sending out reminders for overdue books or generating periodic reports. However, the main capabilities (borrowing, returning, and cataloging) do not require real-time processing.

- Real-Time Aspects: While most LMS activities are not real-time, certain user-facing features may demand soft real-time responsiveness. For example, the search capability should provide results quickly to provide a positive user experience, although this is more about performance optimization than satisfying specific time limitations.

**Concurrency:**

- The Library Management System (LMS) operates in a single-threaded environment.
- Tasks are executed sequentially, one after the other.
- There are no multiple threads involved in concurrent execution.
- Each user request is processed one at a time.
- Processes within the system are not executed in parallel.
- Due to the single-threaded nature of the system, there is no need for thread synchronization techniques.
- The operations of the system are managed by a single thread, ensuring simplicity and straightforwardness in execution.

## 3. Detailed System Design

### 3.1 Static View

The main components of a library management system can be divided into a number of classes, each of which is in charge of particular functions. The UML class diagram and the design's rationale are shown below:

User: Stands in for library patrons who are able to look up, check out, and return books.

Admin: This stands for the administrators who run the library, including the addition of new titles, the maintenance of user accounts, and the supervision of book loans and returns.

Book: Describes the books that are kept in the library, including information on their availability, genre, author, and title.

Issue: Describes the user's loan of a book and includes information about the date of the loan, the due date, and the status of the return.

Request: This is a user's request to borrow a book that isn't available right now.

Return: Indicates that a user is returning a book, along with information on the condition and return date.

Better structure and encapsulation of duties are made possible by separating functions into distinct classes. Because each class is in charge of a certain component of the system, modularity and maintainability are encouraged.

Data encapsulation and information masking are encouraged by the Book class, which contains book-related characteristics and activities. In order to facilitate data encapsulation and information hiding, the User class encapsulates user-related attributes and activities.

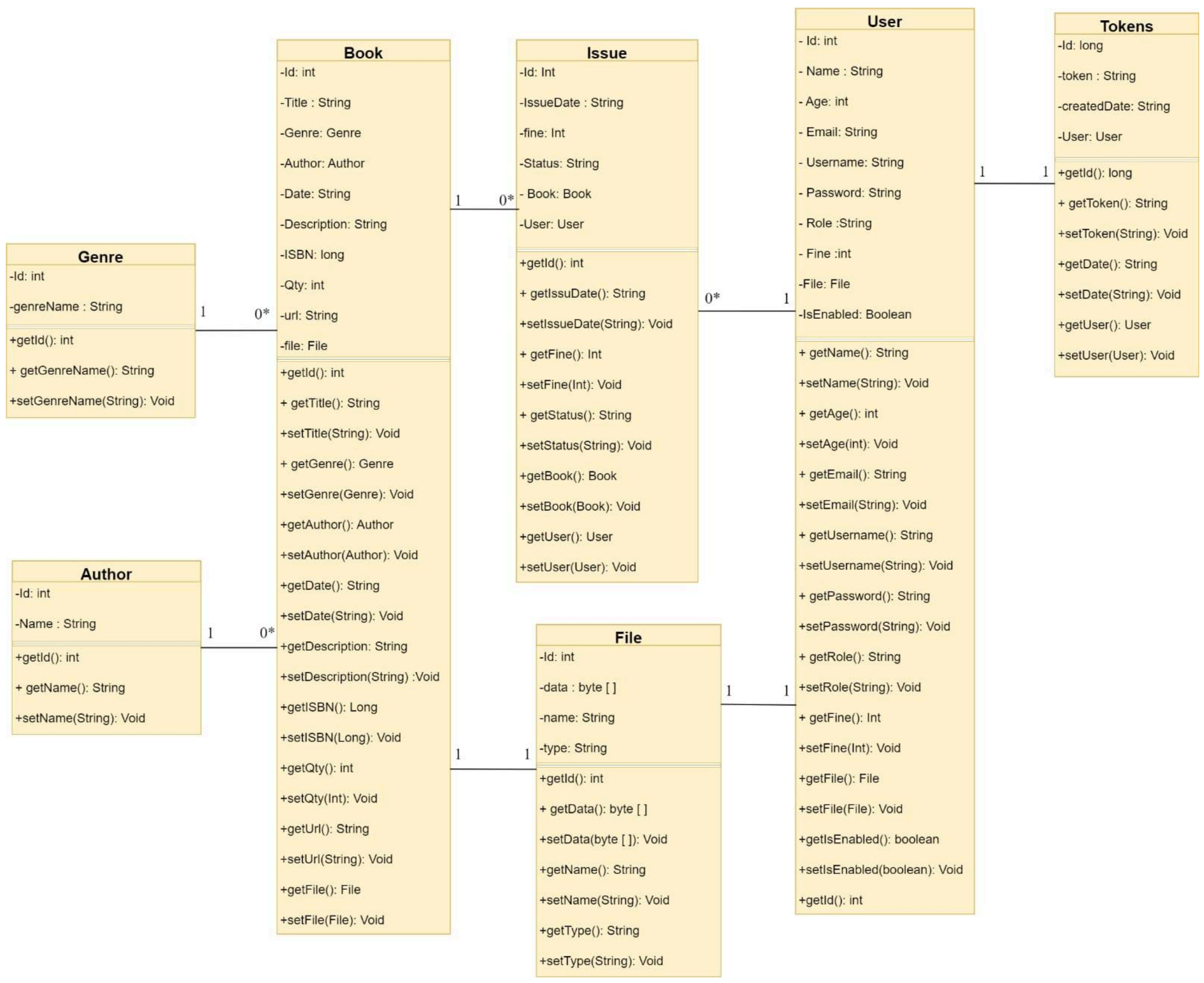


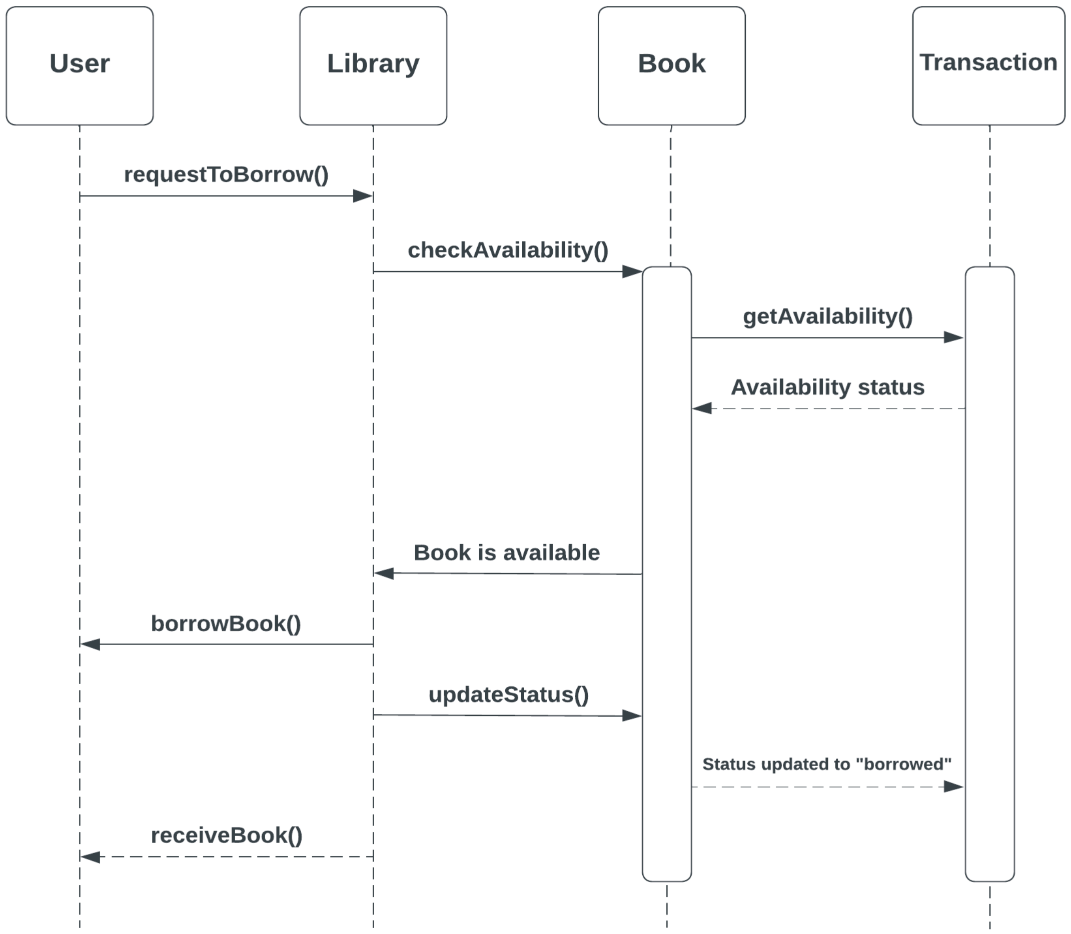
**Fig: UML Class Diagram**

## 3.2 Dynamic View

To provide a dynamic view of the Library Management System (LMS), let's consider a common system function such as borrowing a book:

Sequence Diagram: Borrowing a Book

Actor: User
System: Library Management System (LMS)
Class: User
Class: Library
Class: Book
Class: Transaction



Description:

1. The user initiates a request to borrow a book.
2. The library checks the availability of the requested book.
3. The Library communicates with the Book class to get the availability status.
4. The Book class responds with the availability status.
5. The Library updates the availability status of the book.
6. If the book is available, the Library allows the User to borrow it.
7. The Library updates the status of the book to "borrowed".
8. The User receives the book.

Sprint 1: Implementing Book Borrowing Functionality

Plan:

- Design class diagrams for User, Library, Book, and Transaction.
- Implement basic functionalities such as requesting to borrow a book, checking book availability, and updating book status.
- Create a UML sequence diagram to visualize the borrowing process.

Sprint Review:

- Successfully designed class diagrams for User, Library, Book, and Transaction.
- Implemented core functionalities for borrowing books.
- Created a UML sequence diagram illustrating the borrowing process.
- Identified the need for additional functionalities like returning books in future sprints.

Sprint 2: Implementing Book Returning Functionality

Plan:

- Design and implement functionalities for returning borrowed books.
- Update the UML sequence diagram to include interactions related to returning books.
- Conduct testing and debugging to ensure the smooth functioning of borrowing and returning processes.

Sprint Review:

- Implemented functionalities for returning borrowed books.
- Updated the UML sequence diagram to incorporate interactions related to returning books.
- Conducted thorough testing and debugging to ensure seamless operation of borrowing and returning processes.
- Identified the need for future enhancements such as implementing notifications for overdue books.

Sprint 3: Enhancing User Experience and Performance Optimization

Plan:

- Implement user interface enhancements for a more user-friendly experience.
- Optimize system performance, especially concerning database queries and response times.
- Implement security measures such as user authentication and authorization.

Sprint Review:

- Implemented user interface enhancements to improve the overall user experience.
- Successfully optimized system performance, ensuring efficient operation even under heavy user loads.
- Implemented security measures to safeguard sensitive user and library data.
- Achieved the sprint goals and prepared for further iterations based on user feedback and system requirements.

Additional Items:

Future Enhancements:

- Notifications for Overdue Books: We can add a feature to remind users when their borrowed books are overdue.
- Improved Search: We'll make it easier for users to find books by improving the search function.

Continuous Improvement:

- User Feedback: We'll ask users for their opinions regularly and use their suggestions to make the system better.

Documentation and Training:

- Help Guides: We'll create easy-to-follow guides for library staff to learn how to use the system effectively.

Regular Maintenance:

- Updates and Fixes: We'll make sure to regularly check and update the system to keep it safe and working well for users.

Future Sprints:

- Add a notification system for due dates and overdue books.
- Implement a penalty system for overdue books.

- Explore integration with external systems for eBook lending and online reservations.