# LOW LEVEL DESIGN

Library Management System Web Application

Version<1.0>

# Table of Contents

# 1.Introduction

The purpose of this library management web application is to provide a centralized system for managing the library's collection of books, authors, users, and library staff. The system will allow library staff to easily add, update, and delete books, authors, and other library-related information. It will also allow users to search for books, authors, and other library-related information, and to place holds on books that are currently checked out. The system will also allow library staff to track the borrowing history of patrons and calculate fines if books are returned late.

## 1.1 Purpose:

This LLD document is intended to provide a detailed description of the system's architecture, data structures, algorithms, and interfaces, and to serve as a guide for the development and implementation of the library management web application.

## 1.2 Document Conventions:

**1.2.1 Terminology:** The document uses standard software development terminology, with terms such as entities, repositories, services, and controllers defined and used consistently throughout the document.

**1.2.2 Code snippets:** Code snippets are included throughout the document to provide examples of specific implementation details.

**1.2.3 Diagrams:** diagrams are used to help explain complex concepts and relationships between different parts of the system.

## 1.3 Intended Audience and Reading Suggestions:

This LLD document is intended for developers, project managers and other technical stakeholders who will be involved in the design, development, and implementation of the library management web application.
The document is organized in a way that allows readers to understand the system's architecture and implementation details, even if they are not familiar with all the technologies used.

## 1.4 References:

Spring Boot documentation (https://spring.io/projects/spring-boot)
Hibernate documentation (https://hibernate.org/orm/documentation/5.4/)
MySQL documentation (https://dev.mysql.com/doc/)
JSP documentation (https://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html)

## 2. System Use Cases:

**2.1 Search books:** Users will be able to search for books by title, author, ISBN, or category. The system will return a list of matching books, along with information such as availability and location.

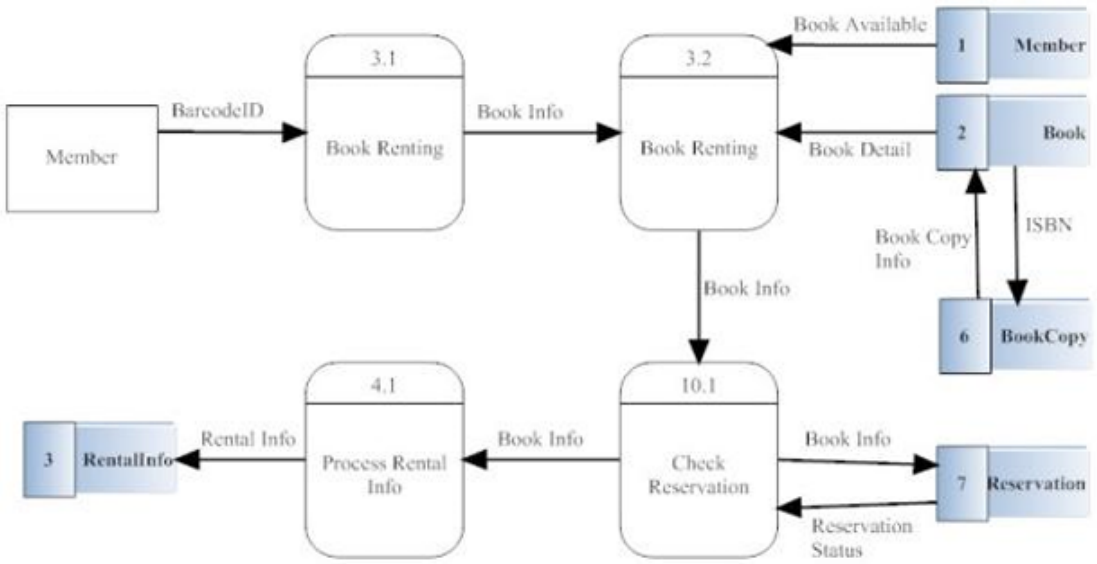



**Fig: Searching or Selecting a Book Flowchart**

**2.2 Place holds**: Users will be able to place holds on books that are currently checked out. The system will notify the user when the book becomes available.

**2.3 Check out books**: Users will be able to check out books from the library. The system will keep track of the due date and calculate fines if books are returned late.

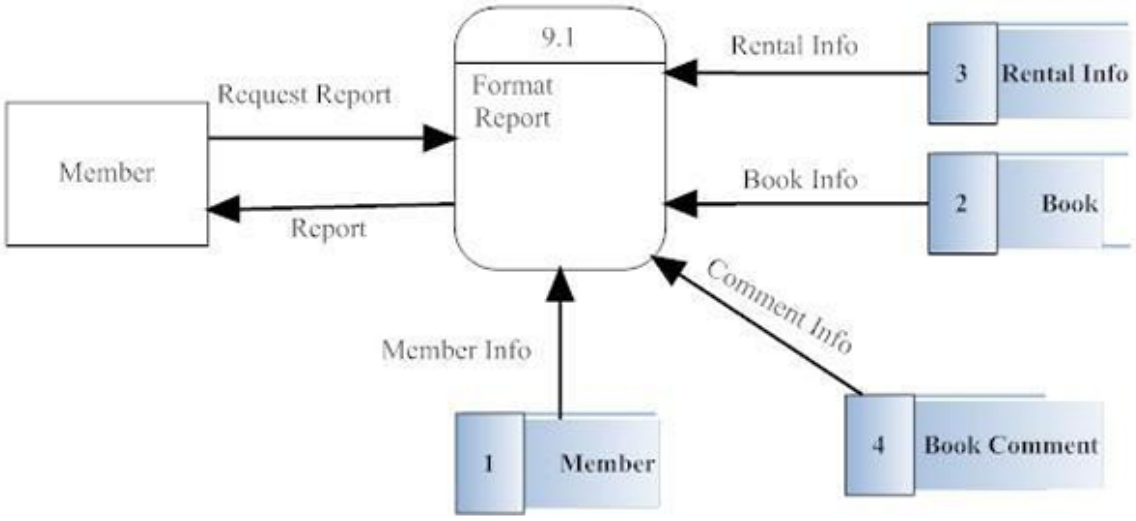



**Fig: Check About The Rental Information of Books**

**2.4 Return books**: Users will be able to return books to the library. The system will update the book's availability and notify any users who have placed holds on the book.

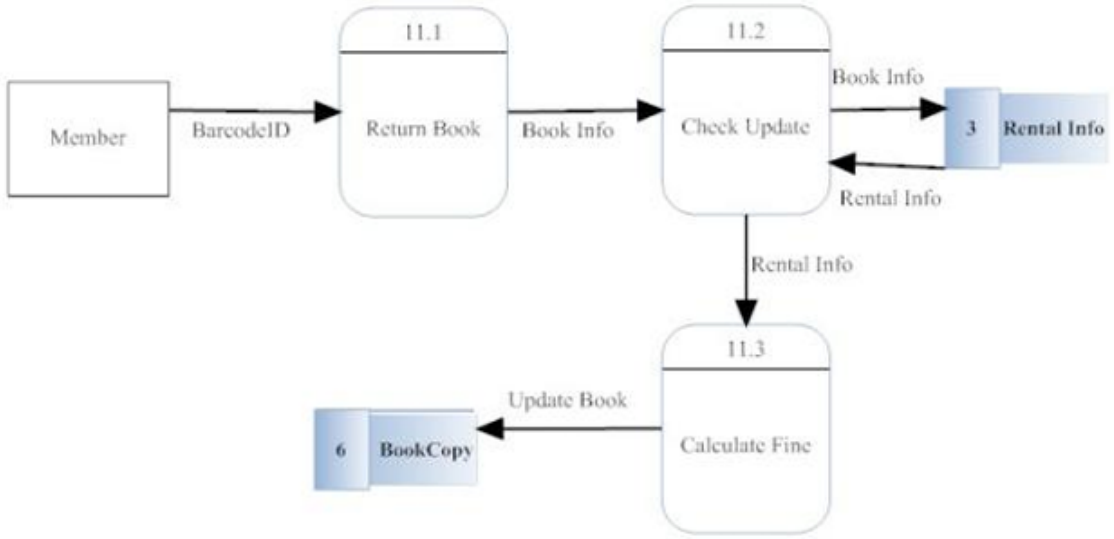



**Fig: Returning a Book FlowChart**

**2.5 Manage library staff**: Library staff will be able to add, update, and delete books, authors, and other library-related information. They will also be able to track the borrowing history of users and calculate fines if books are returned late.

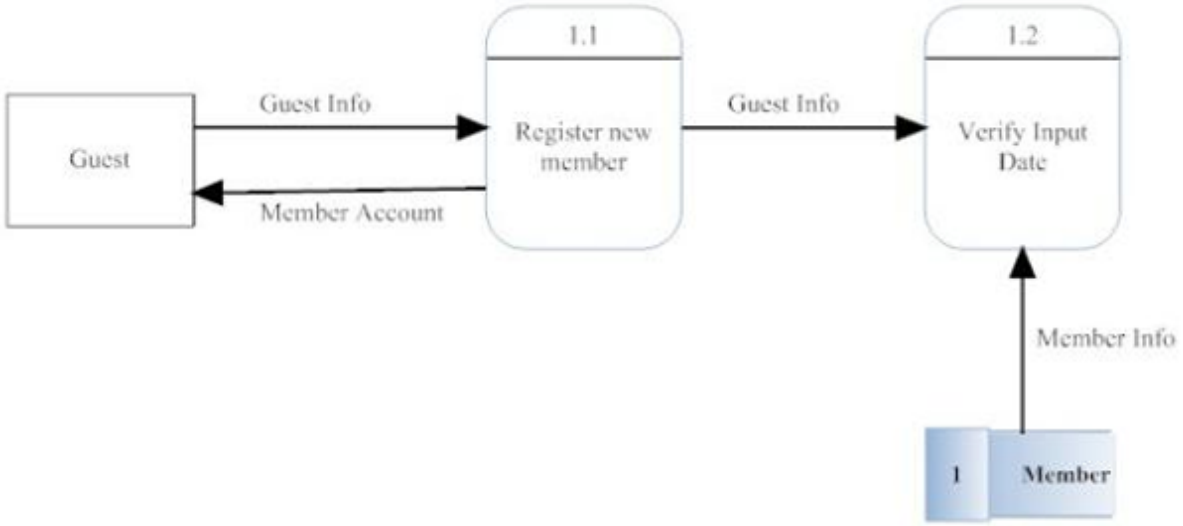



**Fig : User or Member Register Flowchart**

**2.6 Accessibility:** The system will be accessible for users with disabilities and will be compliant with the latest web accessibility standards.

## 3. Detailed System Design

**3.1 Architecture**: The system will be a web-based application with a three-tier architecture: presentation, business, and data access layers. Spring Boot will be used as the foundation for the application, providing the web server, dependency injection, and other features. Hibernate will be used as the Object-Relational Mapping (ORM) framework for interacting with the MySQL database.

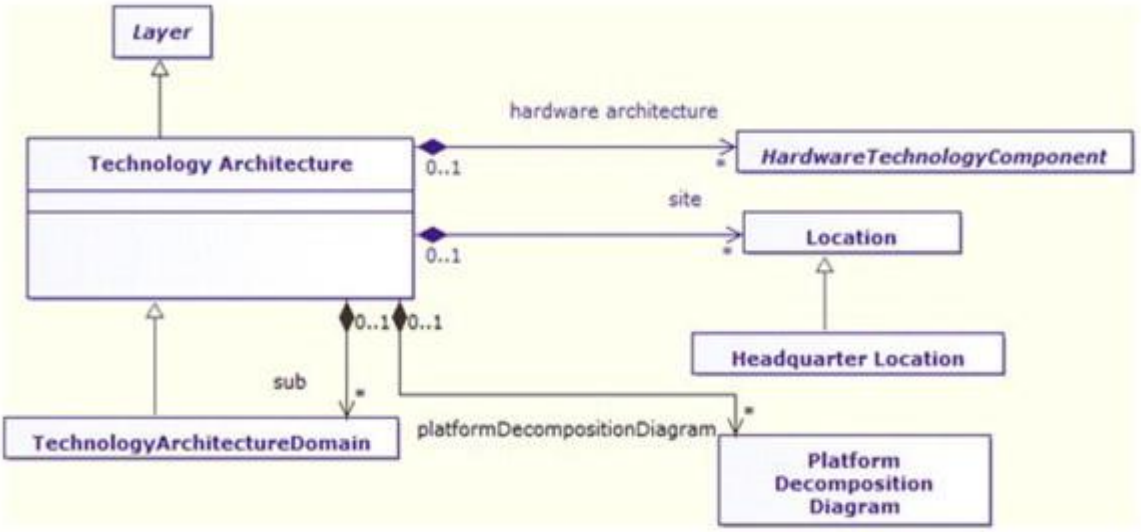



**Fig: Architecture layer**

### 3.1.1 Presentation Layer

It is also known as Client layer. Top most layer of an application. This is the layer we see when we use a software.By using this layer, we can access the web pages. The main functionality of this layer is to communicate with the Application layer. This layer passes the information which is given by the user in terms of keyboard actions, mouse clicks to the Application Layer. For example, the login page of Gmail where an end user could see text boxes and buttons to enter user id,password and to click on sign-in. In simple words, it is to view the application.

An application architecture is a model of interaction between web application components. The kind of architecture for web application strictly depends on the way application logic will be allocated among client and server sides.Technically, it's the skeleton of a web app, including its elements, databases, systems, servers, interfaces, and all the communication happening between them. In more abstract terms, it indicates the logic behind responses to client and server requests.

### 3.1.2 Data Access Layer

The Data-Access Layer (DAL) is a component of a software architecture that is responsible for managing the data storage and retrieval of an application. It sits between the business-logic layer and the data storage system and provides an abstraction layer that allows the business-logic layer to interact with the data storage system without being aware of its specific implementation.

The DAL is responsible for performing tasks such as:

- Connecting to the data storage system and managing the connection.
- Generating and executing SQL queries or other data access commands to retrieve and store data.
- Mapping the data from the data storage system to the application's data objects and vice versa.
- Handling errors and exceptions related to data access.
- Providing support for transactions and other data access features.
- The DAL is designed to be reusable and independent of the business logic and data storage implementation. This allows the application to be easily modified or extended without affecting the underlying data access code.

**3.2 Data Structures:** The system will use a set of entities that will be used to model the various elements of the system, such as books, authors, users, and library staff. The entities will be represented as Java classes, and Hibernate will be used to map them to database tables. The system will use a set of repositories that will be used to interact with the database and perform CRUD operations on the library entities.

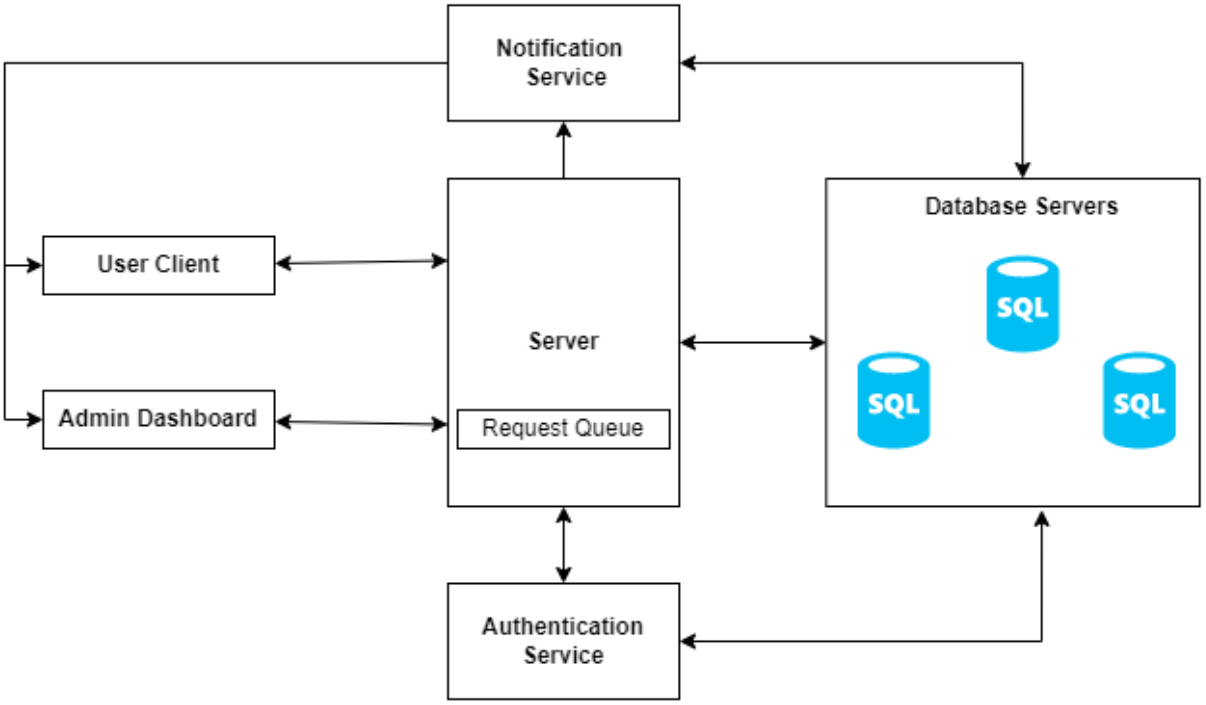
**3.3 Algorithms:** The system will use several algorithms for searching and filtering books, authors, and other library-related information. The system will implement the algorithm to calculate fine if a user returns a book late. The system will implement the algorithm to check the availability of a book before lending it.

**3.4 Interfaces:** The system will use a set of controllers that will handle HTTP requests and route them to the appropriate services. The system will use a set of services that will handle the business logic of the application, such as adding, updating,and deleting books, authors, and other library-related information. The system will use JSP for the front-end, which will provide the user interface for the application.

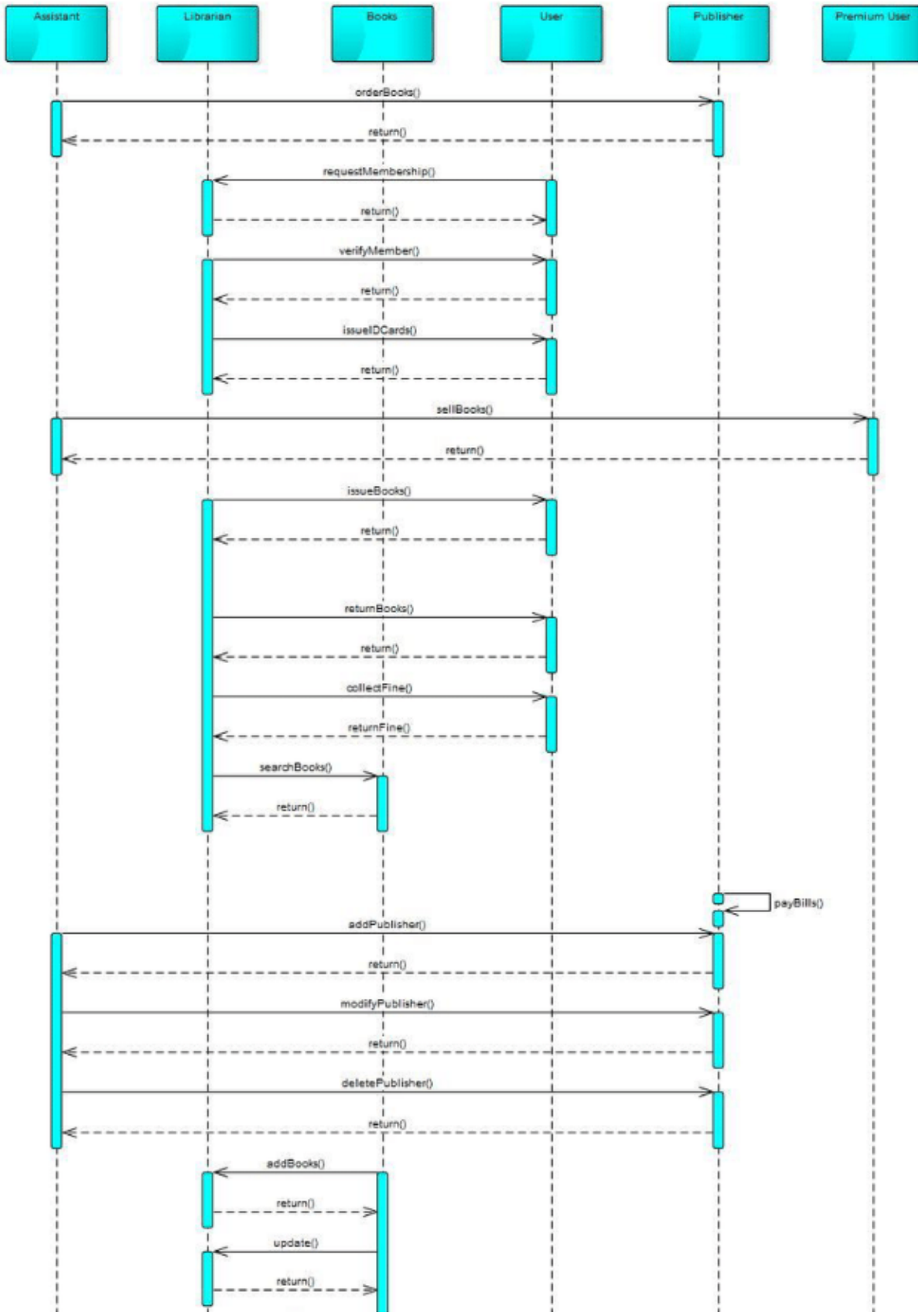
## 3.5 Class Diagram

## 3.6 Use Case Design

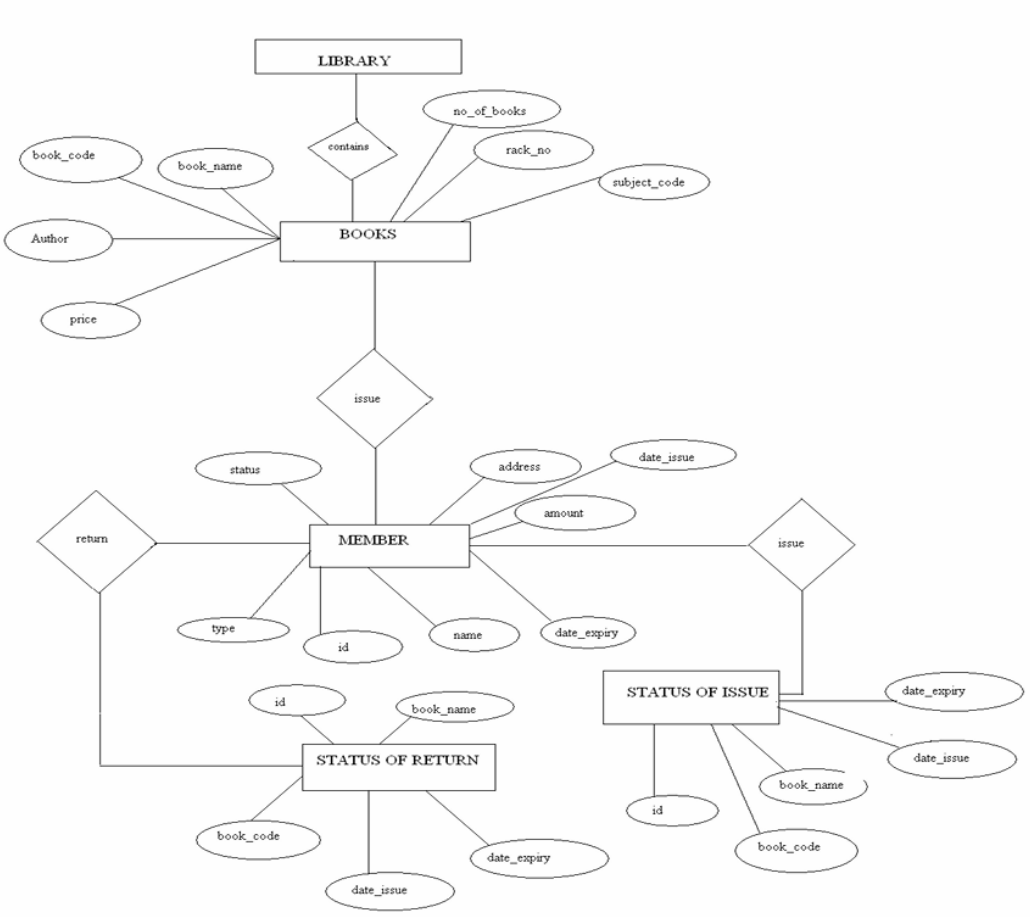
## 3.7 Deployment Model



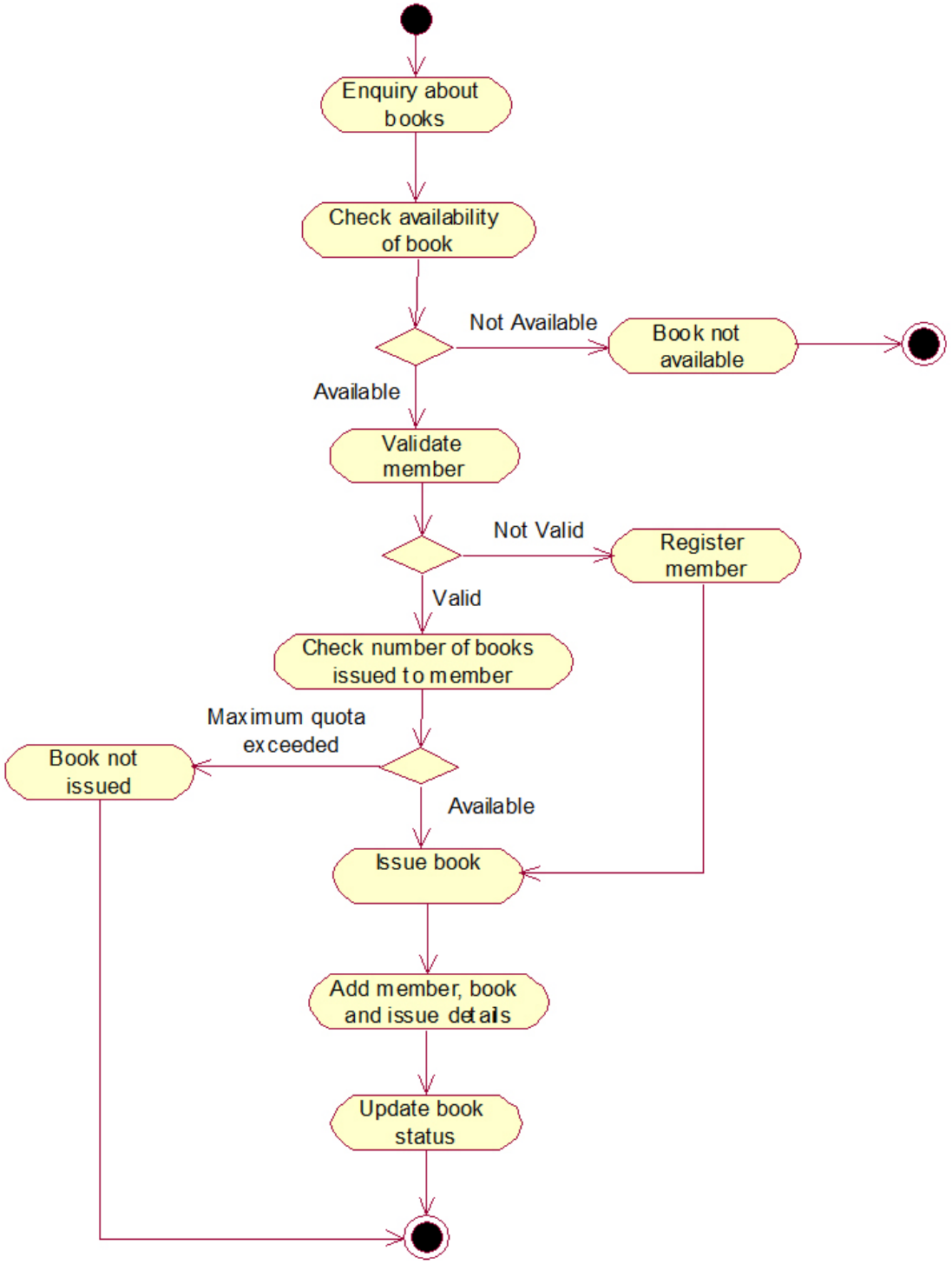
## 3.8 Sequence Diagram



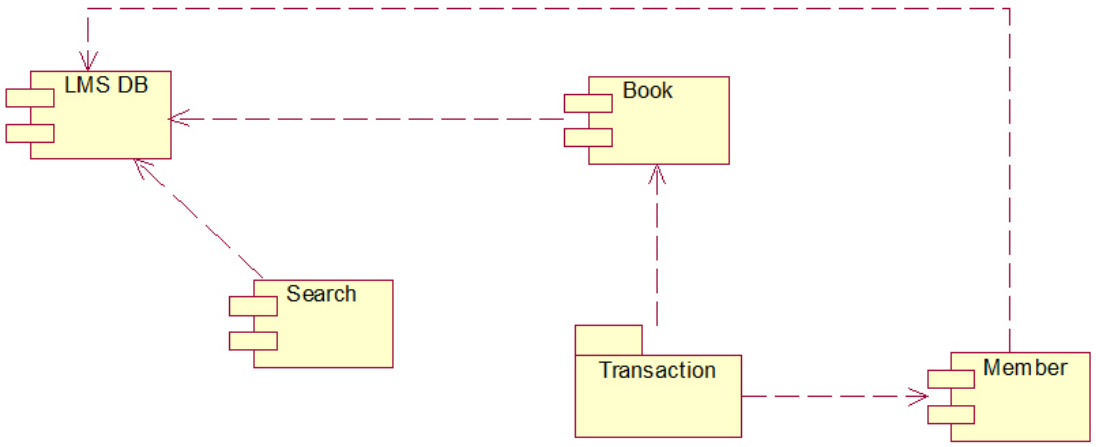
## 3.9 ER Diagram



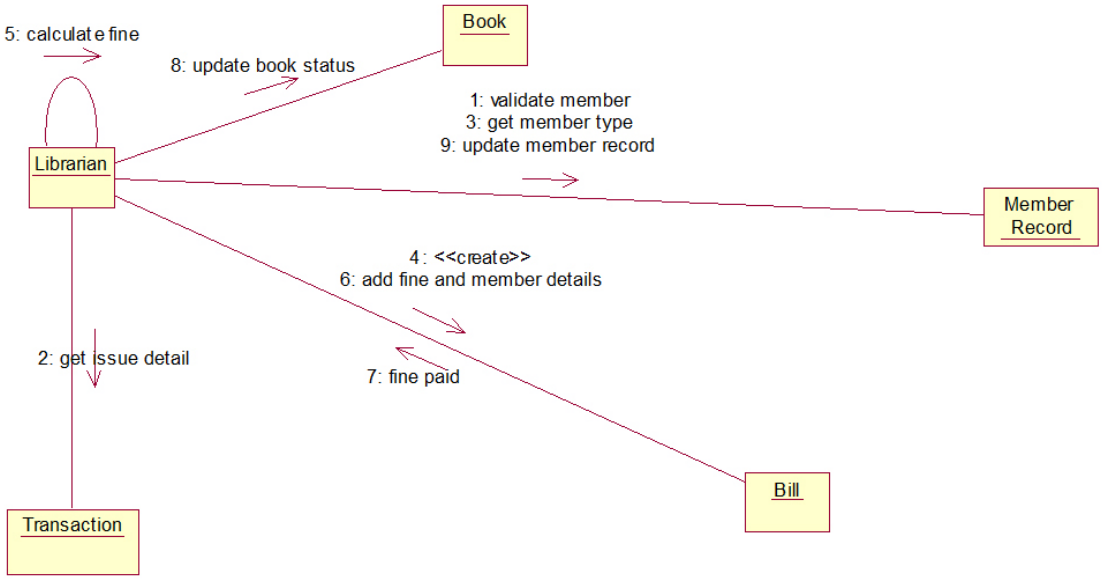
## 3.10 Activity Diagram



## 3.11 Component Diagram




## 3.12 Collaboration Diagram



## 4. Code Snippets

### 4.1 Html for Book Form code

```
<form>
  <label for="title">Title:</label>
  <input type="text" id="title" name="title"><br>
  <label for="author">Author:</label>
  <input type="text" id="author" name="author"><br>
  <label for="isbn">ISBN:</label>
  <input type="text" id="isbn" name="isbn"><br>
  <input type="submit" value="Add Book">
</form>
```

### 4.2 Book Entity Class

```
@Entity
@Table(name = "books")
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private String isbn;
    // Getters and setters here
}
```

### 4.3 Book Repository Class

```
@Repository
public class BookDao {
    @Autowired
    private SessionFactory sessionFactory;
    public void addBook(Book book) {
        sessionFactory.getCurrentSession().save(book);
    }
    public List<Book> getAllBooks() {
        return sessionFactory.getCurrentSession().createQuery("from Book").list();
    }
}
```

### 4.4 Book Config Class

```
@WebServlet("/add-book")
public class AddBookServlet extends HttpServlet {
    @Autowired
    private BookDao bookDao;
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        String title = request.getParameter("title");
        String author = request.getParameter("author");
        String isbn = request.getParameter("isbn");
        Book book = new Book();
        book.setTitle(title);
        book.setAuthor(author);
        book.setIsbn(isbn);
        bookDao.addBook(book);
        response.sendRedirect("book-list.jsp");
    }
}
```

### 4.5 Sql Config (To Create User )

```
CREATE USER 'librarydb_user'@'localhost' IDENTIFIED BY 'spring';

GRANT ALL PRIVILEGES ON C2312_library.* TO
'librarydb_user'@'localhost' WITH GRANT OPTION;

SHOW GRANTS FOR 'librarydb_user'@'localhost';

delimiter ;

DROP SCHEMA IF EXISTS `CS2312_library`;

CREATE SCHEMA `C2312_library` ;

use `C2312_library`;
```

### 4.6 To Create Member Table

```
CREATE TABLE `member` (

`memid` int(15) NOT NULL AUTO_INCREMENT,

`name` varchar(15) NOT NULL,

`address` varchar(15) NOT NULL,

`classification` varchar(15) NOT NULL,

`username` varchar(15) NOT NULL,

`password` varchar(15) NOT NULL,

PRIMARY KEY (`memid`),

UNIQUE KEY `username_UNIQUE` (`username`)

);
```

### 4.7 To Create Book Table

```
CREATE TABLE `book` (
`bookid` int(15) NOT NULL AUTO_INCREMENT,
`title` varchar(15) NOT NULL,
`author` varchar(15) NOT NULL,
`isbn` int(10) NOT NULL,
PRIMARY KEY (`bookid`),
UNIQUE KEY `name_UNIQUE` (`title`)
) ;
```

### 4.8 To Create BookIssue Table

```
CREATE TABLE `bookissue` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`memid` int(15) NOT NULL,
`bookid` int(15) NOT NULL,
`issuedate` date DEFAULT NULL,
PRIMARY KEY (`id`),
FOREIGN KEY (`memid`) references member(memid) ON DELETE CASCADE,
FOREIGN KEY (`bookid`) references book(bookid) ON DELETE CASCADE
);
```

## 4. Security

Security in a library management system is important to protect sensitive information such as personal data of students, financial transactions, and confidential academic documents. Some common security measures that can be implemented in a student library management system include:

User authentication: ensuring that only authorized users can access the system by requiring a login and password.
Data backup: regularly backing up the system's data to protect against loss or damage.
Firewall: implementing a firewall to protect the system from external threats such as hackers.
Regular security updates: keeping the system updated with the latest security patches and fixes to address any known vulnerabilities.

## Appendix

### A Use cases:

Check Out Books:- Users can check out books from the library using the circulation interface.
Renew Books:- Users can renew books that they have checked out from the library.
Place Holds:- Users can place holds on books that are currently checked out.
Pay Fines:- Users can pay fines for overdue books or lost materials using the user account management interface.
Search Catalog:- Users can search the library's catalog to find books and other materials.
Manage User Accounts:- Library staff can manage user accounts, including adding new users and updating existing user information.
Manage Library Collection:- Library staff can manage the library's collection, including adding new books and removing outdated materials.

## Glossary

HTTP : HyperText Transfer Protocol

SQL : Structured Query Language

User :a person chosen, named, or honored as a special guardian, protector, or supporter.

Publisher : They are Nothing But Author’s..

LMS DB : Library Management System DataBase

Dao: Data Access Object ,which can also be called as Repository

DAL : Data-Access Layer in Architecture

User ID: A unique identifier for each library User.

First Name: The first name of the library user.

Last Name: The last name of the library user.

Address: The postal address of the library user.

Phone Number: The phone number of the library user.

Email Address: The email address of the library user.

Library Card Number: A unique identifier for each library card.

Book ID: A unique identifier for each book in the library's collection.

Title: The title of the book.

Author: The author of the book.

ISBN: The International Standard Book Number (ISBN) of the book.

Call Number: The call number used to locate the book in the library.

Check Out Date: The date when the book was checked out by a user.

Due Date: The date when the book is due to be returned to the library.

Check In Date: The date when the book was checked in to the library.

Fine Amount: The amount of any fines due for an overdue book.