

ITE 1942 – ICT PROJECT

PROJECT REPORT Level 01

Library Management System

Submitted by:

GHSK WIJESOORIYA

E2245624

Bachelor of Information Technology (External Degree)

Faculty of Information Technology

University of Moratuwa

Table of Contents

1.INTRODUCTION	5
2.RELATED WORK	7
3.SYSTEMS ANALYSIS	8
4.SYSTEM DESIGN.....	9
5.SYSTEM IMPLEMENTATION	10
6.APPENDIX	11
7.REFERENCES	12

1. INTRODUCTION

1.1 Background & Motivation

In the digital age, libraries are essential for providing access to information and resources. However, traditional library management methods often lead to inefficiencies, such as difficulties in tracking inventory, managing user accounts, and processing transactions. The need for a robust Library Management System (LMS) has become increasingly apparent, as it can significantly enhance the operational efficiency of libraries.

The impact of these problems is profound, affecting librarians, users, and the overall functionality of the library. Users may experience frustration due to long wait times for book availability, while librarians may struggle with manual record-keeping and inventory management. Existing solutions, while available, often lack the comprehensive features necessary to address the unique needs of different libraries.

1.2 Problem in Brief

The primary problem addressed by this project is the inefficiency in managing library resources, which includes tracking book inventory, managing user accounts, and processing transactions. The current manual systems are prone to errors and can lead to lost books and frustrated users.

To solve this problem, the proposed solution is to develop a Library Management System that automates these processes, providing a user-friendly interface for both librarians and users, thereby improving overall efficiency and user satisfaction.

1.3 Aims & Objectives

The aim of this project is to develop a system for addressing the inefficiencies in library management. The specific objectives to achieve this aim include:

- ◆ Conducting a critical analysis of system requirements to understand the needs of users and librarians.
- ◆ Designing and developing a comprehensive Library Management System that includes features such as user registration, book search, borrowing and returning books, and fine calculation.
- ◆ Evaluating the proposed system through user testing and feedback to ensure it meets the needs of its users

1.4 Summary

This chapter introduced the problem of inefficient library management and outlined the motivation for developing a computer-based solution. The aims and objectives of the project were also discussed. The next chapter will review related work in the field of library management systems, highlighting existing solutions and their limitations.

2. RELATED WORK

In this chapter, we will explore existing library management systems and their features. We will analyze their strengths and weaknesses, providing a context for the development of our system.

2.1 Existing Solutions

In the realm of library management, several systems have been developed to address the challenges faced by libraries in managing their resources. This section reviews some of the most prominent existing solutions, highlighting their features, strengths, and limitations.

2.1.1 LibraryThing

Overview: LibraryThing is a web-based application that allows users to catalog their personal libraries. It is primarily designed for individual users rather than institutional libraries.

Features:

- Users can create a personal catalog of their books.
- Offers social networking features, allowing users to connect with other book lovers.
- Provides recommendations based on user preferences.

Strengths:

- User-friendly interface that is easy to navigate.
- Strong community features that encourage user interaction and engagement.
- Extensive database of books, making it easy to find and catalog titles.

Limitations:

- Lacks comprehensive features for managing library transactions, such as checkouts and returns.
- Not designed for institutional use, making it unsuitable for libraries that require robust management capabilities.
- Limited reporting and analytics features for tracking library usage and inventory.

2.1.2 Koha

Overview: Koha is an open-source Integrated Library System (ILS) that provides a comprehensive suite of tools for library management. It is widely used by libraries around the world.

Features:

- Cataloging and inventory management.
- User account management, including registration and borrowing history.
- Circulation management, including checkouts, returns, and fines.
- Reporting tools for analyzing library usage and inventory.

Strengths:

- Highly customizable due to its open-source nature, allowing libraries to tailor the system to their specific needs.
- A large community of users and developers contributes to ongoing improvements and support.
- Comprehensive features that cover all aspects of library management.

Limitations:

- The complexity of the system may be overwhelming for smaller libraries or those with limited technical expertise.
- Requires ongoing maintenance and updates, which may necessitate dedicated IT resources.
- User interface may not be as intuitive as some commercial solutions, leading to a steeper learning curve for staff.

2.1.3 Alma

Overview: Alma is a cloud-based library management solution developed by Ex Libris. It integrates various library functions into a single platform, making it suitable for academic and research libraries.

Features:

- Unified management of print, electronic, and digital resources.
- Advanced analytics and reporting capabilities.
- Integration with discovery tools and learning management systems.

Strengths:

- Comprehensive features that support a wide range of library functions, including acquisitions, cataloging, and circulation.
- Cloud-based infrastructure allows for easy access and scalability.
- Strong support for electronic resource management.

Limitations:

- High cost of implementation and subscription, which may be prohibitive for smaller libraries.
- Complexity of the system may require extensive training for staff.
- Dependence on internet connectivity for access to the system.

2.1.4 Libsys

Overview: Libsys is a library management software that caters to various types of libraries, including academic, public, and special libraries.

Features:

- Cataloging, circulation, and acquisition modules.
- User management and reporting tools.
- Support for multiple languages and regional settings.

Strengths:

- Flexible and scalable, making it suitable for libraries of all sizes.
- Comprehensive support and training services provided by the vendor.
- User-friendly interface that simplifies library operations.

Limitations:

- Licensing costs can be high, especially for smaller libraries.
- Some users report that customer support can be slow to respond.
- Limited customization options compared to open-source solutions.

2.1.5 OpenBiblio

Overview: OpenBiblio is an open-source library management system designed for small to medium-sized libraries.

Features:

- Basic cataloging and circulation functionalities.
- User account management and reporting features.
- Web-based interface for easy access.

Strengths:

- Free to use, making it an attractive option for budget-conscious libraries.
- Simple and straightforward interface that is easy to navigate.
- Active community support for troubleshooting and enhancements.

Limitations:

- Limited features compared to more comprehensive systems like Koha or Alma.
- May not be suitable for larger libraries with complex needs.
- Requires some technical knowledge for installation and maintenance.

Summary of Existing Solutions

The existing library management systems each offer unique features and capabilities, catering to different types of libraries and user needs. While some systems excel in user engagement and community features, others provide comprehensive management tools for institutional libraries. However, many existing solutions face limitations in terms of complexity, cost, and customization. The proposed Library Management System aims to address these gaps by providing a user-friendly, efficient, and cost-effective solution tailored to

3. Systems Analysis

This chapter provides a comprehensive analysis of the requirements for the Library Management System (LMS). It includes both functional and non-functional requirements, ensuring that all aspects of the system are thoroughly addressed. The requirements outlined in this chapter will guide the design and implementation of the system.

3.1 System Requirements

The system requirements for the Library Management System can be categorized into hardware, software, and user requirements.

3.1.1 Hardware Requirements

Server Specifications:

- o **Processor:** A minimum of a dual-core processor (e.g., Intel Xeon or AMD Ryzen) to handle multiple requests efficiently.
- o **RAM:** At least 16 GB of RAM to ensure smooth operation, especially during peak usage times.
- o **Storage:** A minimum of 1 TB HDD or SSD for data storage, allowing ample space for book records, user data, and transaction logs.
- o **Network Interface:** Gigabit Ethernet for fast data transfer and connectivity.

Client Machines:

- o **Specifications:** Computers or tablets with at least 8 GB of RAM, a modern web browser (Chrome, Firefox, or Edge), and a screen resolution of at least 1366x768 pixels for optimal user experience.

- o **Peripherals:** Printers for printing receipts and reports, barcode scanners for efficient book checkouts and returns.

Network Infrastructure:

- o **Internet Connection:** A stable broadband connection with a minimum speed of 20 Mbps for optimal performance, especially if the system is cloud-based.
- o **Router:** A reliable router to manage internal network traffic and ensure connectivity between client machines and the server.

3.1.2 Software Requirements

Operating System: Debian 12 (Bookworm) as the server operating system, chosen for its stability, security, and extensive package management capabilities.

Database Management System: MySQL or PostgreSQL for data storage, providing robust support for relational data and complex queries.

Web Server: Apache or Nginx to host the web application, ensuring efficient handling of HTTP requests and serving web pages to users.

Programming Languages:

- o **Backend:** PHP or Python for server-side logic, chosen for their ease of use and extensive libraries for web development.
- o **Frontend:** HTML5, CSS3, and JavaScript for creating a responsive and interactive user interface.

Development Tools: Integrated Development Environment (IDE) such as Visual Studio Code or PyCharm for coding and debugging.

3.1.3 User Requirements

Librarians: Require access to manage books, users, and transactions, including the ability to generate reports and handle user inquiries.

Library Users: Need a user-friendly interface to search for books, manage their accounts, view borrowing history, and receive notifications about due dates and fines.

Administrators: Require access to system settings, user management, and reporting features, along with the ability to perform system maintenance and updates.

3.2 Functional Requirements

Functional requirements define the specific behaviors and functions of the Library Management System. The following are the key functional requirements identified for the LMS:

3.2.1 User Management

User Registration: The system shall allow new users to register by providing personal details such as name, email, contact number, and address. Validation checks shall ensure that all required fields are completed and that the email format is correct.

User Authentication: The system shall authenticate users through a secure login process using a username and password. Passwords shall be hashed and stored securely to protect user data.

User Roles: The system shall support different user roles (e.g., librarian, user, administrator) with varying access levels. Each role shall have specific permissions, such as the ability to add or delete books, manage user accounts, and generate reports.

3.2.2 Book Management

Add Books: The system shall allow librarians to add new books to the inventory, including details such as title, author, ISBN, publication year, category, and number of copies available. The system shall validate the ISBN format and check for duplicates before adding.

Update Books: The system shall enable librarians to update existing book information, including changing the status of a book (e.g., available, checked out, lost).

Delete Books: The system shall allow librarians to remove books from the inventory, ensuring that any associated transactions are handled appropriately.

3.2.3 Search Functionality

Search Books: The system shall provide a search feature that allows users to find books by title, author, ISBN, or category. The search results shall display relevant information, including

4. SYSTEM DESIGN

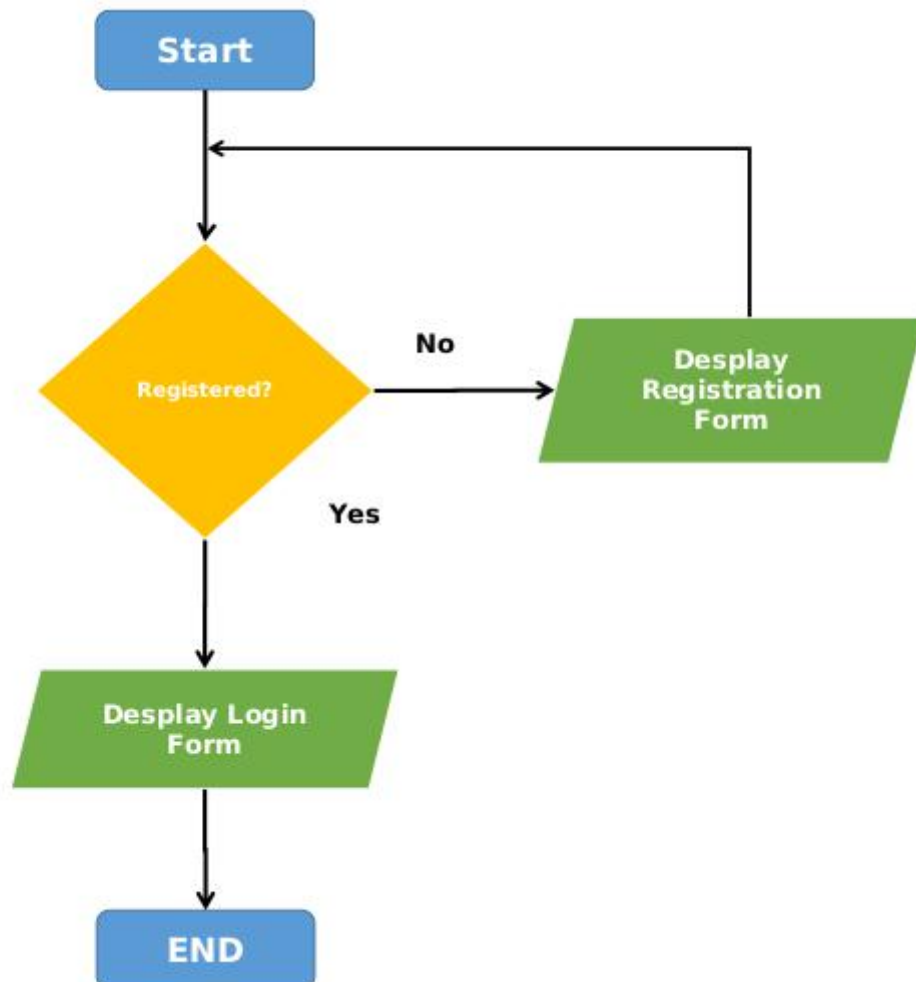
This chapter outlines the design of the Library Management System (LMS), detailing how each functional requirement is modeled using flowcharts. The flowcharts provide a visual representation of the processes involved in user login, book borrowing, and book return, illustrating the sequence of actions and decisions.

4.1 Flow Charts

Flowcharts are essential for understanding the workflow of the system. Below are the flowcharts for the key functionalities of the LMS.

4.1.1 User Login Flowchart

Description: This flowchart illustrates the process of user login in the LMS.

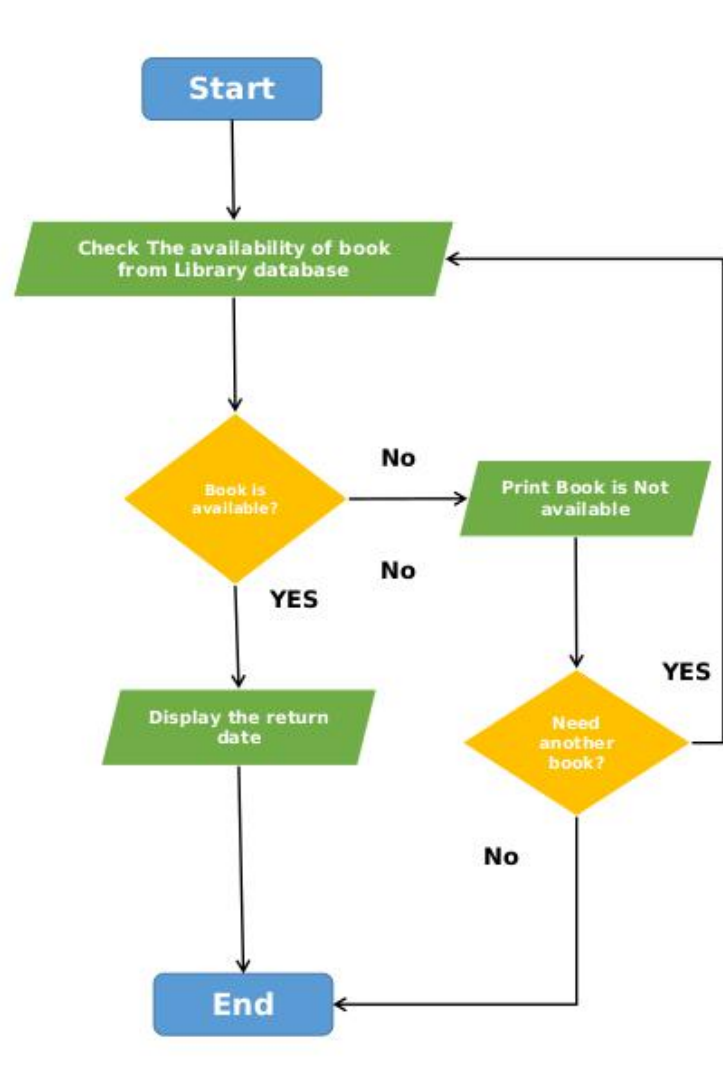


Process Steps:

1. **Start:** The user initiates the login process.
2. **Input Credentials:** The user enters their username and password.
3. **Validate Credentials:** The system checks if the username and password match the records.
 1. **Yes:** Proceed to the next step.
 2. **No:** Arrow pointing to a rectangle labeled "Notify user: Invalid credentials" and then back to "Input Credentials".
4. **Access Granted:** The system grants access to the user dashboard.
5. **End:** The login process is complete.

4.1.2 Book Borrowing Flowchart

Description: This flowchart outlines the process for borrowing a book from the library.

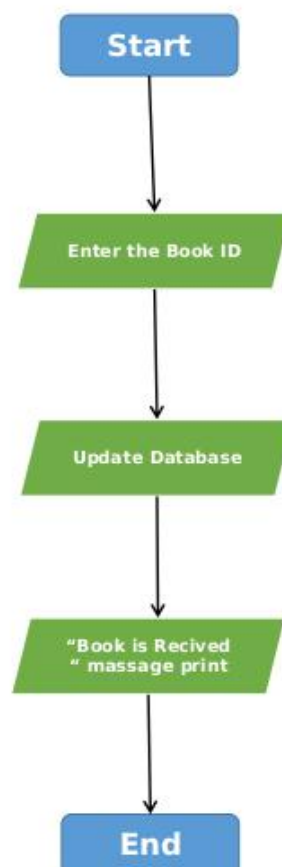


Process Steps:

1. **Start:** The user initiates the book borrowing process.
2. **Search for Book:** The user searches for the desired book using the title, author, or ISBN.
3. **Check Book Availability:** The system checks if the book is available for borrowing.
 1. **Yes:** Proceed to the next step.
 2. **No:** Arrow pointing to a rectangle labeled "Notify user: Book not available" and then back to "Search for Book".
4. **Confirm Borrowing:** The user confirms the borrowing action.
5. **Update Records:** The system updates the user's borrowing history and the book's status to "checked out".
6. **End:** The book borrowing process is complete.

4.1.3 Book Return Flowchart

Description: This flowchart depicts the process of returning a borrowed book.



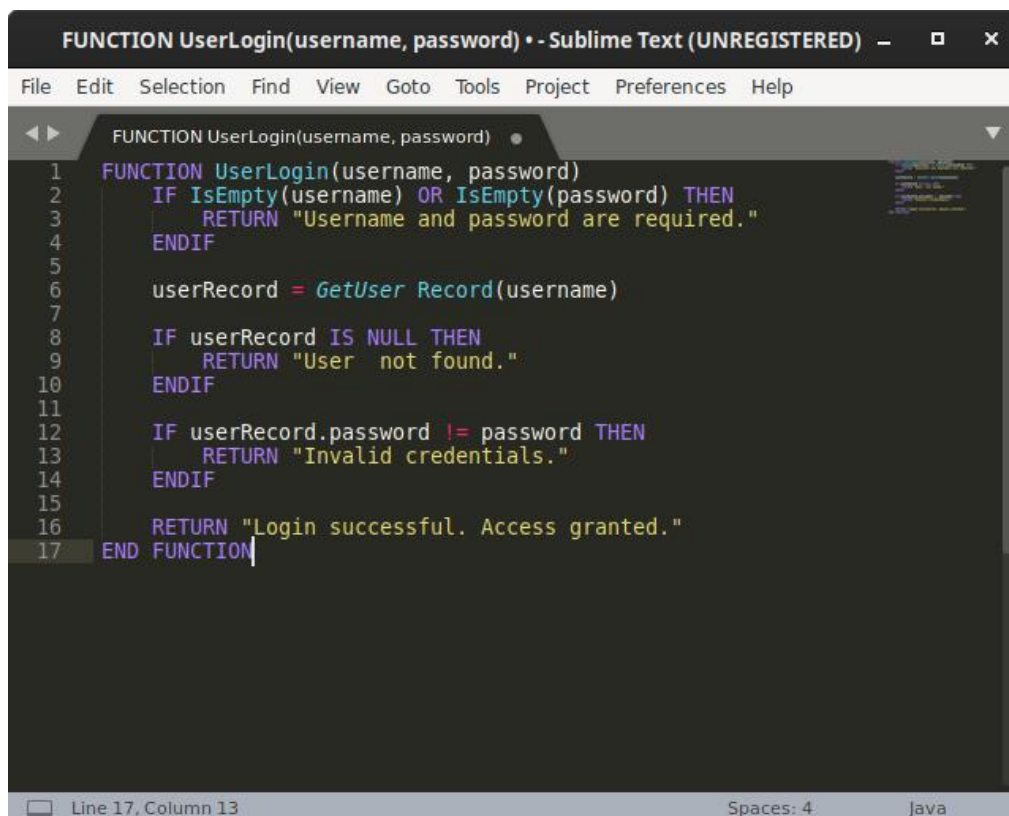
Process Steps:

1. **Start:** The user initiates the book return process.
2. **Input Book Details:** The user enters the book ID or scans the book's barcode.
3. **Validate Book:** The system checks if the book is in the user's borrowing history.
 1. **Yes:** Proceed to the next step.
 2. **No:** Arrow pointing to a rectangle labeled "Notify user: Book not found in borrowing history" and then back to "Input Book Details".
4. **Update Records:** The system updates the user's borrowing history and the book's status to "available".
5. **End:** The book return process is complete.

4.2 Pseudocodes

Pseudocode provides a high-level description of the algorithms used in the system. Each pseudocode corresponds to a specific functional requirement.

4.2.1 User Login Pseudocode



```
FUNCTION UserLogin(username, password) • - Sublime Text (UNREGISTERED) - □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
FUNCTION UserLogin(username, password)
1 FUNCTION UserLogin(username, password)
2   IF IsEmpty(username) OR IsEmpty(password) THEN
3     RETURN "Username and password are required."
4   ENDIF
5
6   userRecord = GetUserRecord(username)
7
8   IF userRecord IS NULL THEN
9     RETURN "User not found."
10  ENDIF
11
12  IF userRecord.password != password THEN
13    RETURN "Invalid credentials."
14  ENDIF
15
16  RETURN "Login successful. Access granted."
17 END FUNCTION
```

Line 17, Column 13 Spaces: 4 java

Explanation:

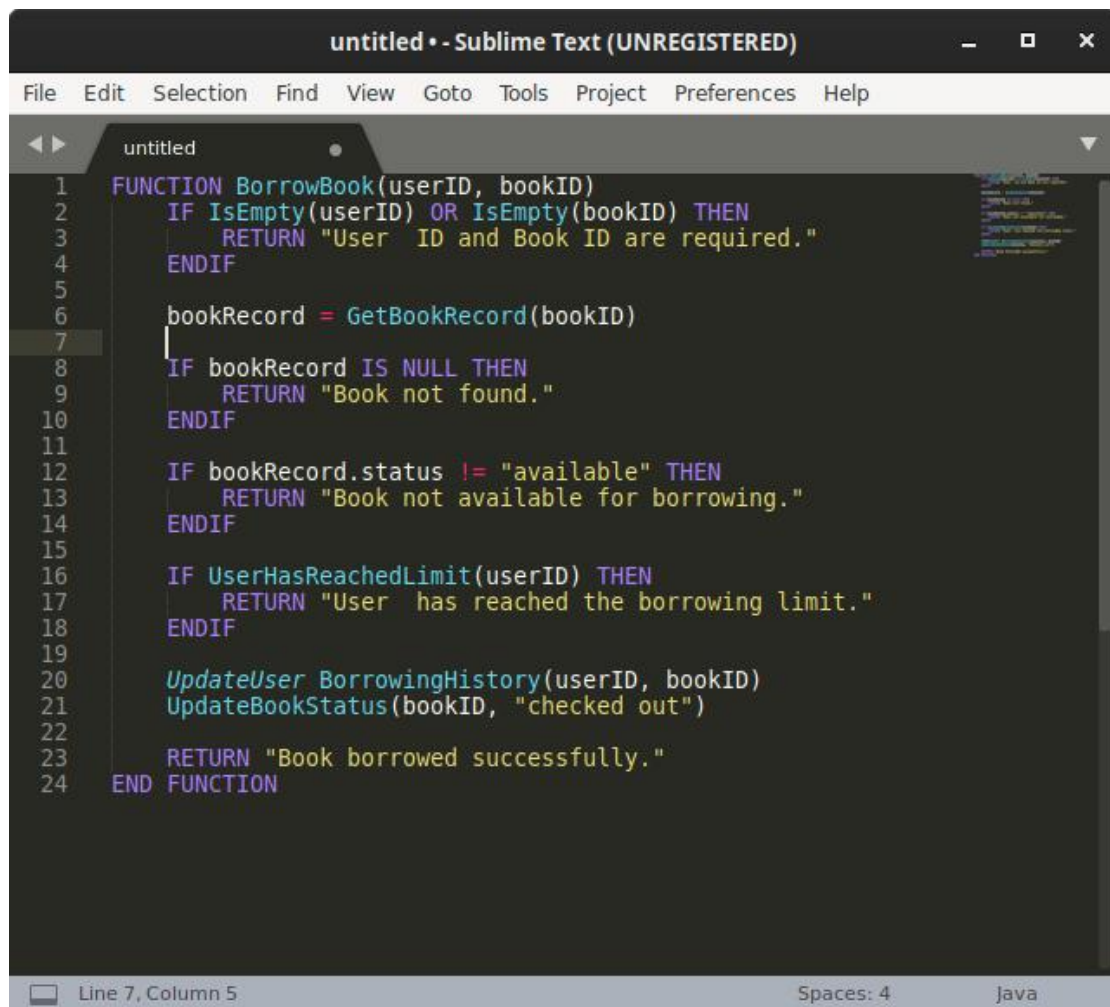
The function checks if the username and password fields are empty.

It retrieves the user record based on the username.

If the user is found, it checks if the password matches.

It returns appropriate messages based on the validation results.

4.2.2 Book Borrowing Pseudocode



```
untitled • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

1 FUNCTION BorrowBook(userID, bookID)
2   IF IsEmpty(userID) OR IsEmpty(bookID) THEN
3     RETURN "User ID and Book ID are required."
4   ENDIF
5
6   bookRecord = GetBookRecord(bookID)
7
8   IF bookRecord IS NULL THEN
9     RETURN "Book not found."
10  ENDIF
11
12  IF bookRecord.status != "available" THEN
13    RETURN "Book not available for borrowing."
14  ENDIF
15
16  IF UserHasReachedLimit(userID) THEN
17    RETURN "User has reached the borrowing limit."
18  ENDIF
19
20  UpdateUserBorrowingHistory(userID, bookID)
21  UpdateBookStatus(bookID, "checked out")
22
23  RETURN "Book borrowed successfully."
24 END FUNCTION

Line 7, Column 5 Spaces: 4 java
```

Explanation:

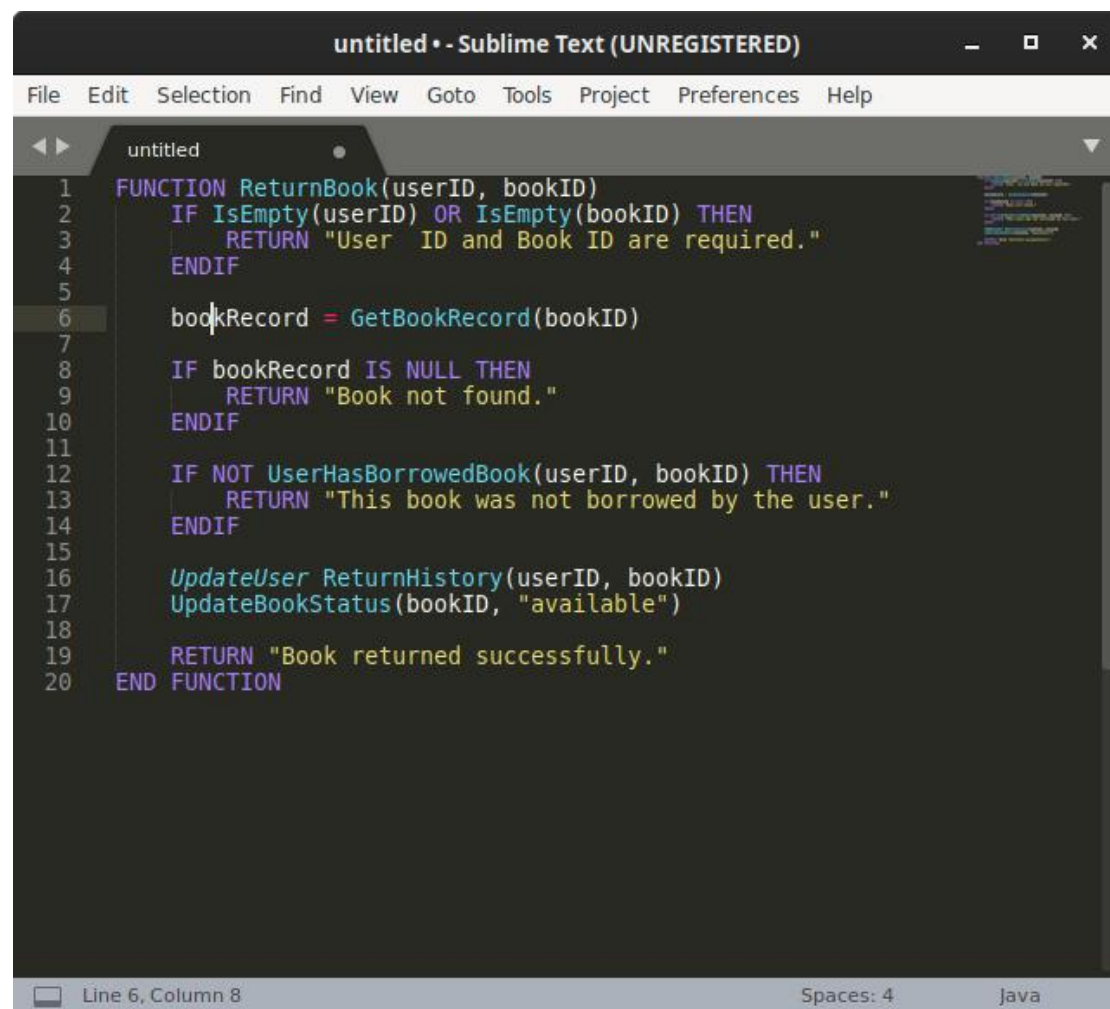
The function checks if the user ID and book ID are provided.

It retrieves the book record based on the book ID.

It checks if the book is available and if the user has reached their borrowing limit.

If all checks pass, it updates the user's borrowing history and the book's status.

4.2.3 Book Return Pseudocode



```
untitled • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

1 FUNCTION ReturnBook(userID, bookID)
2   IF IsEmpty(userID) OR IsEmpty(bookID) THEN
3     RETURN "User ID and Book ID are required."
4   ENDIF
5
6   bookRecord = GetBookRecord(bookID)
7
8   IF bookRecord IS NULL THEN
9     RETURN "Book not found."
10  ENDIF
11
12  IF NOT UserHasBorrowedBook(userID, bookID) THEN
13    RETURN "This book was not borrowed by the user."
14  ENDIF
15
16  UpdateUser ReturnHistory(userID, bookID)
17  UpdateBookStatus(bookID, "available")
18
19  RETURN "Book returned successfully."
20 END FUNCTION

Line 6, Column 8 Spaces: 4 Java
```

Explanation:

The function checks if the user ID and book ID are provided.

It retrieves the book record based on the book ID.

It checks if the user has borrowed the book.

If all checks pass, it updates the user's return history and the book's status.

4.3 Summary

In this section, we have provided pseudocode for the core functionalities of the Library Management System, including user login, book borrowing, and book return processes. Each pseudocode outlines the logical steps and conditions necessary to implement these functionalities, serving as a guide for the development phase of the project.

5. SYSTEM IMPLEMENTATION

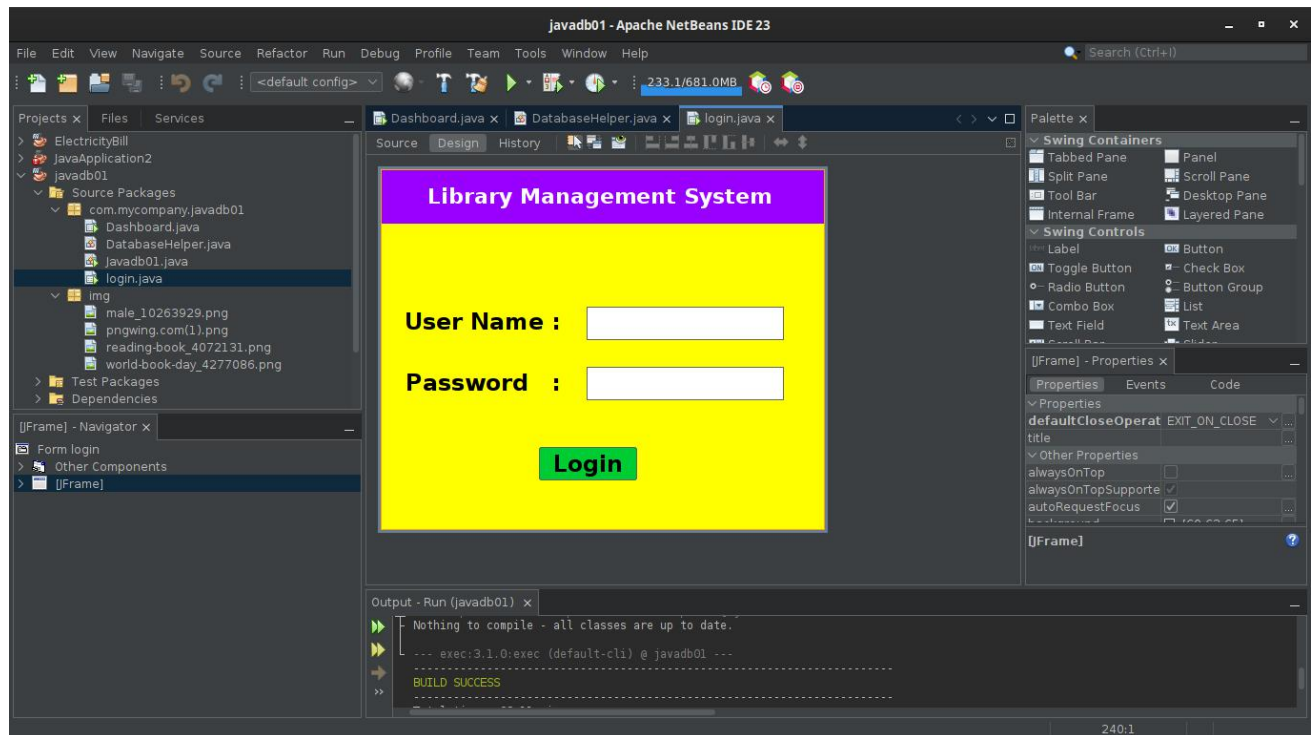
This section outlines the implementation of the Library Management System (LMS) using Java for the backend and MySQL for the database. It includes user interfaces, selected code snippets, and descriptions of the major processes implemented in the system.

5.1 User Interfaces

The user interfaces of the LMS are designed using Java Swing for desktop applications. Below are a few selected user interfaces that represent the core functionalities of the system.

5.1.1 Login Interface

Description: The login interface allows users to enter their credentials to access the system.



```
package com.mycompany.javadb01;

import javax.swing.JOptionPane;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 */
```

```
private boolean isValidLogin(String username, String password) {
    String query = "SELECT * FROM Admin WHERE Username = ? AND Password = ?";
    try (Connection conn = DatabaseHelper.connect();
        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, username);
        stmt.setString(2, password);
        ResultSet rs = stmt.executeQuery();

        return rs.next(); // returns true if credentials are valid

    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        return false;
    }
}
```

```

private void LoginBtnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String username = UserName.getText().trim();
    String password = Password.getText().trim();

    if (username.isEmpty() || password.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Please enter both username and password.", "Input Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

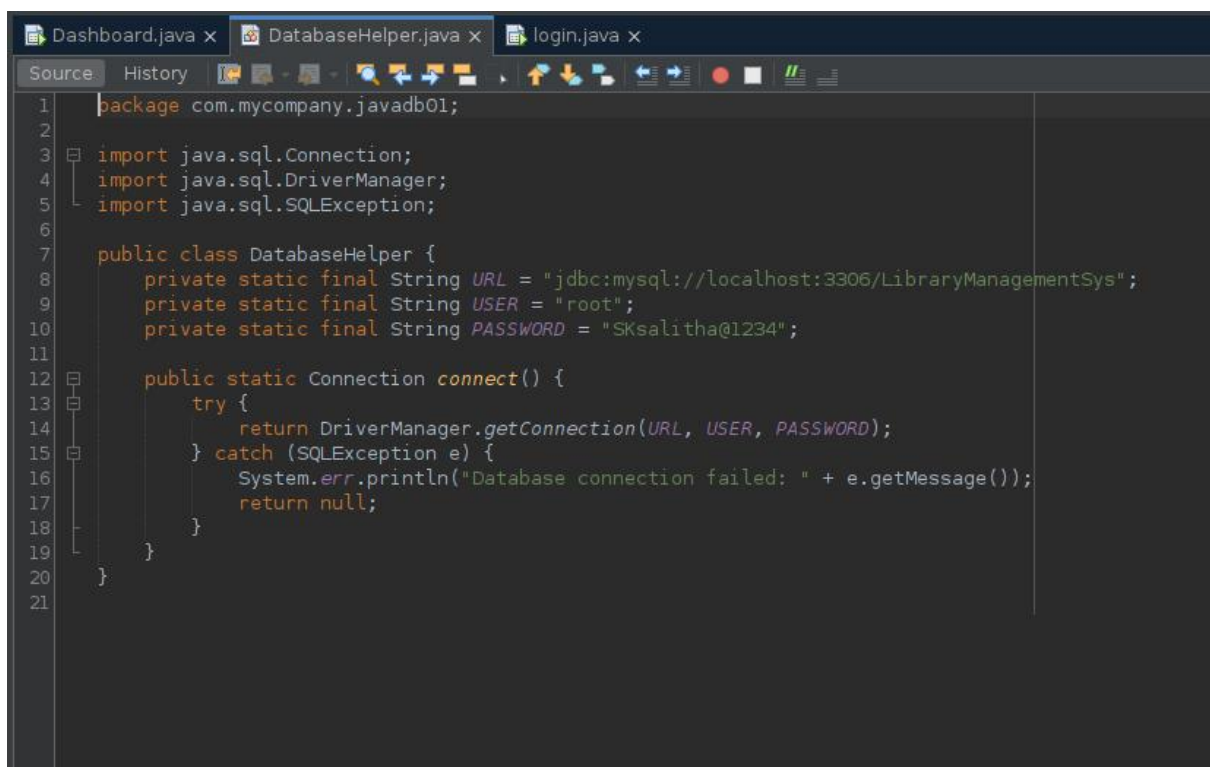
    boolean loginSuccessful = isValidLogin(username, password);

    if (loginSuccessful) {
        // JOptionPane.showMessageDialog(this, "Login successful!", "Welcome", JOptionPane.INFORMATION_MESSAGE);
        new Dashboard().setVisible(true); // Open Dashboard
        this.dispose(); // Close the login form
    } else {
        JOptionPane.showMessageDialog(this, "Invalid username or password.", "Login Failed", JOptionPane.ERROR_MESSAGE);
    }
}
}

```

5.2 Database Connection Class

This class handles the connection to the MySQL database.



```

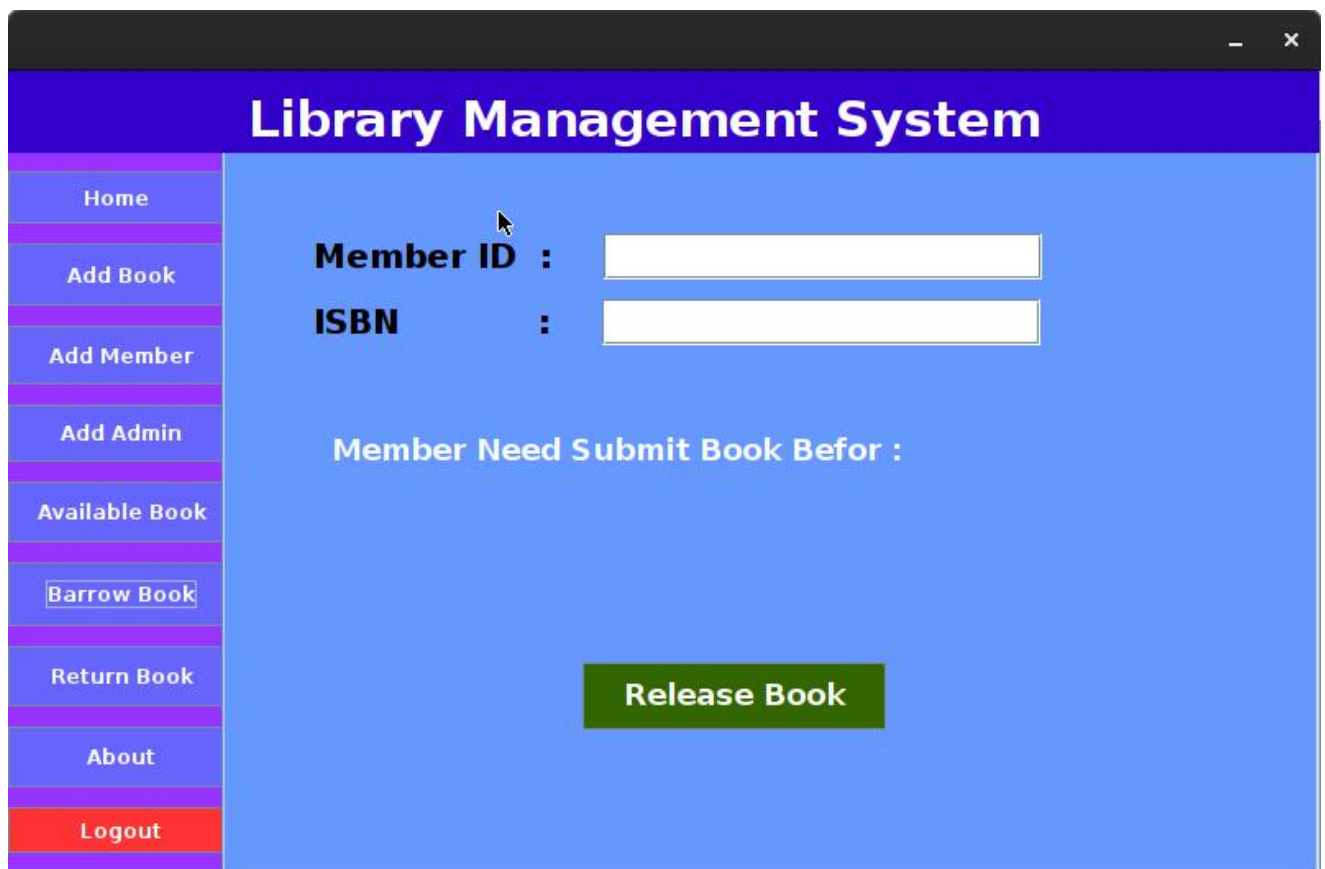
1 package com.mycompany.javadb01;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class DatabaseHelper {
8     private static final String URL = "jdbc:mysql://localhost:3306/LibraryManagementSys";
9     private static final String USER = "root";
10    private static final String PASSWORD = "SKsalitha@1234";
11
12    public static Connection connect() {
13        try {
14            return DriverManager.getConnection(URL, USER, PASSWORD);
15        } catch (SQLException e) {
16            System.err.println("Database connection failed: " + e.getMessage());
17            return null;
18        }
19    }
20 }
21

```

5.3 Selected Code Snippets

This section includes selected code snippets that demonstrate the implementation of key functionalities in the LMS.

5.3.1 Book Borrowing Implementation



The screenshot displays the 'Library Management System' web application. On the left is a vertical sidebar with navigation links: Home, Add Book, Add Member, Add Admin, Available Book, Barrow Book (highlighted), Return Book, About, and Logout. The main content area has a light blue background. At the top, it says 'Library Management System'. Below this, there are two input fields for 'Member ID' and 'ISBN', each preceded by a colon. A mouse cursor is positioned over the 'Member ID' field. Below the input fields, the text 'Member Need Submit Book Befor :' is displayed. At the bottom center, there is a green button labeled 'Release Book'.

Library Management System	
Home	<p>Member ID : <input type="text"/></p> <p>ISBN : <input type="text"/></p> <p>Member Need Submit Book Befor :</p> <p>Release Book</p>
Add Book	
Add Member	
Add Admin	
Available Book	
Barrow Book	
Return Book	
About	
Logout	

```

private void borrowBook() throws SQLException {

    String memberIdStr = BrrowMemberID.getText().trim();

    String isbn = BarrowBookISBN.getText().trim();


    // Validate input

    if (memberIdStr.isEmpty() || isbn.isEmpty()) {

        JOptionPane.showMessageDialog(this, "Please enter both Member ID and
ISBN.", "Input Error", JOptionPane.ERROR_MESSAGE);

        return;

    }


    int memberId;

    try {

        memberId = Integer.parseInt(memberIdStr);

    } catch (NumberFormatException e) {

        JOptionPane.showMessageDialog(this, "Member ID must be a number.",
"Input Error", JOptionPane.ERROR_MESSAGE);

        return;

    }


    // Use a single connection for the whole process

    try (Connection conn = DatabaseHelper.connect()) {

        conn.setAutoCommit(false); // Begin transaction


        // Check if Member_ID exists


        String memberQuery = "SELECT * FROM Members WHERE Member_ID
= ?";

```



```

        try (PreparedStatement memberStmt =
conn.prepareStatement(memberQuery)) {

            memberStmt.setInt(1, memberId);

            ResultSet memberRs = memberStmt.executeQuery();

            if (!memberRs.next()) {

                JOptionPane.showMessageDialog(this, "Wrong Member ID", "Error",
JOptionPane.ERROR_MESSAGE);

                conn.rollback();

                return;

            }

        }

```

// Check if the book is available

```

String bookQuery = "SELECT * FROM Books WHERE ISBN = ?";

int bookId;

int quantity;

String status;

try (PreparedStatement bookStmt = conn.prepareStatement(bookQuery)) {

    bookStmt.setString(1, isbn);

    ResultSet bookRs = bookStmt.executeQuery();

    if (!bookRs.next()) {

        JOptionPane.showMessageDialog(this, "Book not found", "Error",
JOptionPane.ERROR_MESSAGE);

        conn.rollback();

        return;

```

```

    }

    bookId = bookRs.getInt("Book_ID");

    quantity = bookRs.getInt("Quantity");

    status = bookRs.getString("Status");

    if (quantity <= 0 || status.equals("Issued")) {

        JOptionPane.showMessageDialog(this, "Book is currently not available",
        "Error", JOptionPane.ERROR_MESSAGE);

        conn.rollback();

        return;

    }

}

// Proceed to borrow the book

String transactionQuery = "INSERT INTO Transactions (Book_ID,
Member_ID, Issue_Date) VALUES (?, ?, ?)";

try (PreparedStatement transactionStmt =
conn.prepareStatement(transactionQuery)) {

    Date issueDate = new Date(System.currentTimeMillis());

    Date returnDate = new Date(System.currentTimeMillis() + 7 * 24 * 60 * 60
* 1000); // 1 week later

    // double fine = 0.00; // Assuming no fine at the time of borrowing

    transactionStmt.setInt(1, bookId);

    transactionStmt.setInt(2, memberId);

    transactionStmt.setDate(3, issueDate);

    // transactionStmt.setDate(4, returnDate);

```

```

// transactionStmt.setDouble(4, fine);

transactionStmt.executeUpdate();


// Update return date label

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

BookReturnDate.setText(sdf.format(returnDate));

}


// Update the quantity and status of the book

String updateBookQuery = "UPDATE Books SET Quantity = ?, Status = ?
WHERE Book_ID = ?";

try (PreparedStatement updateBookStmt =
conn.prepareStatement(updateBookQuery)) {

    int newQuantity = quantity - 1;

    String newStatus = (newQuantity == 0) ? "Not Available" : "Available";

    updateBookStmt.setInt(1, newQuantity);

    updateBookStmt.setString(2, newStatus);

    updateBookStmt.setInt(3, bookId);

    updateBookStmt.executeUpdate();

}


// Commit transaction

conn.commit();


// Show success message

```

```
JOptionPane.showMessageDialog(this, "Book borrowed successfully! Please  
return it by " + BookReturnDate.getText(), "Success",  
JOptionPane.INFORMATION_MESSAGE);
```

```
BrrowMemberID.setText("");
```

```
BarrowBookISBN.setText("");
```

```
BookReturnDate.setText("YYYY/MM/DD");
```

```
BrrowMemberID.requestFocus();
```

```
} catch (SQLException e) {
```

```
// Rollback transaction in case of error
```

```
try (Connection conn = DatabaseHelper.connect()) {
```

```
    conn.rollback();
```

```
} catch (SQLException rollbackEx) {
```

```
    JOptionPane.showMessageDialog(this, "Failed to rollback transaction: " +  
rollbackEx.getMessage(), "Rollback Error", JOptionPane.ERROR_MESSAGE);
```

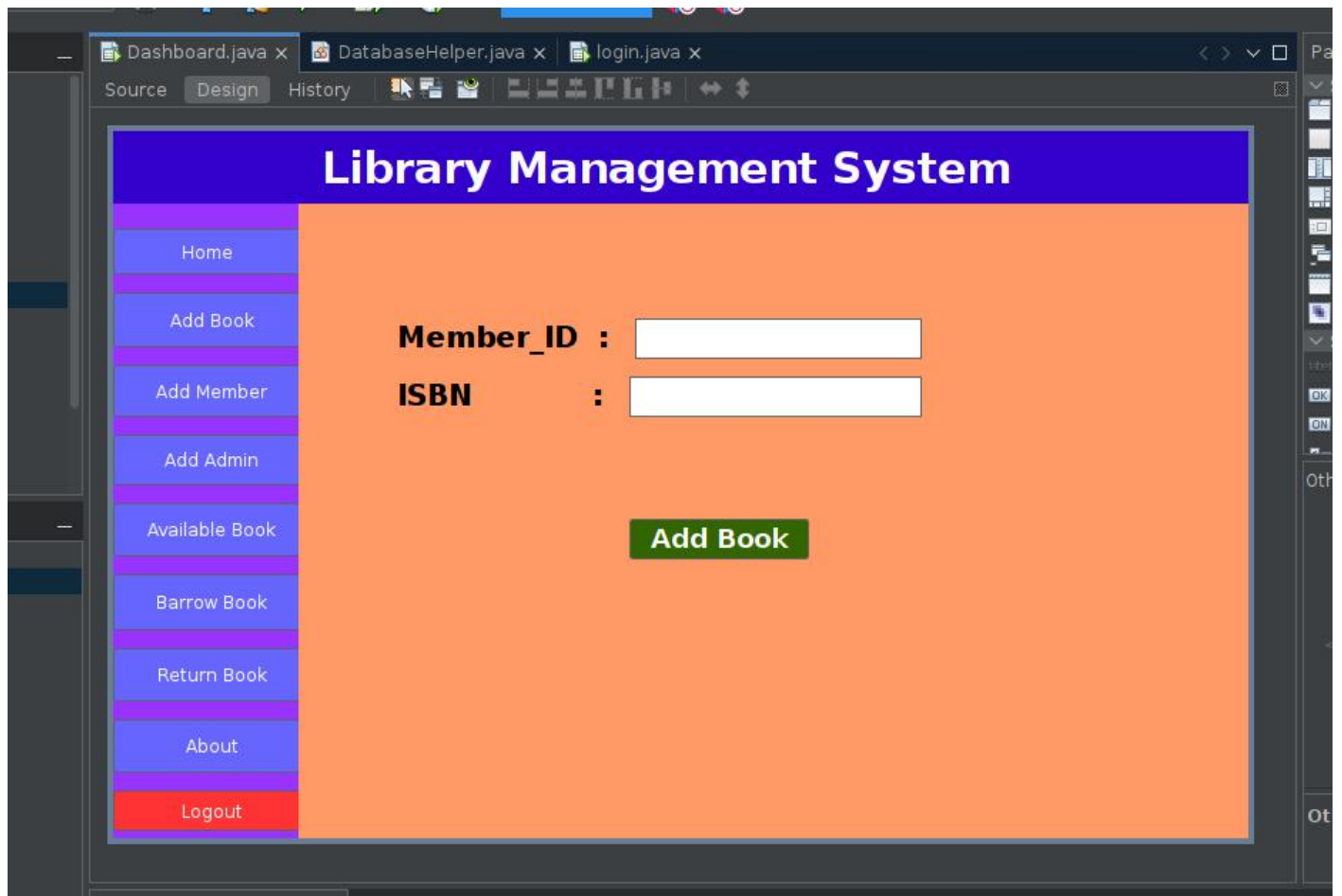
```
}
```

```
JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(),  
"Error", JOptionPane.ERROR_MESSAGE);
```

```
}
```

```
}
```

5.3.2 Book Returning Implementation



```
private void returnBook() {  
    String memberIdStr = ReturnMemberID.getText().trim();  
    String isbn = ReturnBookISBN.getText().trim();  
  
    // Validate input  
    if (memberIdStr.isEmpty() || isbn.isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Please enter both Member ID and ISBN.", "Input  
Error", JOptionPane.ERROR_MESSAGE);  
        return;  
    }  
  
    int memberId;  
    try {  
        memberId = Integer.parseInt(memberIdStr);
```

```

    } catch (NumberFormatException e) {

        JOptionPane.showMessageDialog(this, "Member ID must be a number.", "Input Error",
JOptionPane.ERROR_MESSAGE);

        return;
    }

    try (Connection conn = DatabaseHelper.connect()) {

        if (conn == null) {

            JOptionPane.showMessageDialog(this, "Failed to connect to the database.", "Connection
Error", JOptionPane.ERROR_MESSAGE);

            return;
        }

        conn.setAutoCommit(false); // Begin transaction

        // Check if the transaction exists

        String transactionQuery = "SELECT * FROM Transactions WHERE Member_ID = ?
AND Book_ID = (SELECT Book_ID FROM Books WHERE ISBN = ?)";

        try (PreparedStatement transactionStmt = conn.prepareStatement(transactionQuery)) {

            transactionStmt.setInt(1, memberId);

            transactionStmt.setString(2, isbn);

            ResultSet transactionRs = transactionStmt.executeQuery();

            if (!transactionRs.next()) {

                JOptionPane.showMessageDialog(this, "No transaction found for this Member ID and
ISBN.", "Error", JOptionPane.ERROR_MESSAGE);

                conn.rollback();

                return;
            }

            // Retrieve the issue date

            Date issueDate = transactionRs.getDate("Issue_Date");

            Date currentDate = new Date(System.currentTimeMillis());

            // Calculate the due date (assuming a 14-day loan period)

```

```

        long dueDateMillis = issueDate.getTime() + (14 * 24 * 60 * 60 * 1000); // Add 14 days in
        milliseconds

        Date dueDate = new Date(dueDateMillis); // Convert back to java.sql.Date

        // Calculate fine if the book is returned late

        long diffInMillies = currentDate.getTime() - dueDate.getTime();

        long daysLate = diffInMillies / (1000 * 60 * 60 * 24); // Convert milliseconds to days

        double fine = (daysLate > 0) ? daysLate * 10 : 0; // RS 10 per day

        // Update the book's quantity and status

        String bookQuery = "UPDATE Books SET Quantity = Quantity + 1, Status = CASE
        WHEN Quantity + 1 = 1 THEN 'Available' ELSE Status END WHERE ISBN = ?";

        try (PreparedStatement bookStmt = conn.prepareStatement(bookQuery)) {

            bookStmt.setString(1, isbn);

            bookStmt.executeUpdate();

        }

        // Update the return date and fine in the Transactions table

        String updateTransactionQuery = "UPDATE Transactions SET Return_Date = ?, Fine = ?
        WHERE Member_ID = ? AND Book_ID = (SELECT Book_ID FROM Books WHERE ISBN
        = ?)";

        try (PreparedStatement updateTransactionStmt =
        conn.prepareStatement(updateTransactionQuery)) {

            java.sql.Date returnDate = new java.sql.Date(System.currentTimeMillis()); // Correctly
            create java.sql.Date

            updateTransactionStmt.setDate(1, returnDate);

            updateTransactionStmt.setDouble(2, fine);

            updateTransactionStmt.setInt(3, memberId);

            updateTransactionStmt.setString(4, isbn);

            updateTransactionStmt.executeUpdate();

        }

        // Commit transaction

        conn.commit();

        // Show success message with fine information

```

```

        String fineMessage = (fine > 0) ? " You have a fine of RS " + fine + " for late return." :
        "",

        JOptionPane.showMessageDialog(this, "Book returned successfully!" + fineMessage,
        "Success", JOptionPane.INFORMATION_MESSAGE);

        // Clear input fields

        ReturnMemberID.setText("");

        ReturnBookISBN.setText("");

    } catch (SQLException e) {

        // Rollback transaction in case of error

        conn.rollback();

        JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error",
        JOptionPane.ERROR_MESSAGE);

    }

    } catch (SQLException e) {

        JOptionPane.showMessageDialog(this, "Database connection error: " + e.getMessage(),
        "Error", JOptionPane.ERROR_MESSAGE);

    }

}

```


5.3.3 Add Member Implementation

The screenshot shows a web application window titled "Library Management System". On the left is a vertical sidebar with navigation links: Home, Add Book, Add Member (highlighted), Add Admin, Available Book, Barrow Book, Return Book, About, and Logout. The main content area has a yellow background and contains a registration form with the following fields: First Name, Last Name, Address, Phone No, and Email. Each field is a text input box. Below the form is a green "Register" button.

```
private void addMember() {  
  
    String firstName = MemberFirstName.getText().trim();  
    String lastName = MemberLastName.getText().trim();  
    String address = MemberAddress.getText().trim();  
    String phone = MemberPhone.getText().trim();  
    String email = MemberEmail.getText().trim();  
  
    if (firstName.isEmpty() || lastName.isEmpty() || address.isEmpty() || phone.isEmpty() ||  
email.isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Please fill in all fields.", "Input Error",  
JOptionPane.ERROR_MESSAGE);  
        return;  
    }  
  
    // SQL Query to insert member
```

```
String query = "INSERT INTO Members (First_Name, Last_Name, Address, Phone, Email)
VALUES (?, ?, ?, ?, ?)";
```

```
try (Connection conn = DatabaseHelper.connect();
    PreparedStatement stmt = conn.prepareStatement(query)) {
    stmt.setString(1, firstName);
    stmt.setString(2, lastName);
    stmt.setString(3, address);
    stmt.setString(4, phone);
    stmt.setString(5, email);
    stmt.executeUpdate();

    JOptionPane.showMessageDialog(this, "Member added successfully!", "Success",
JOptionPane.INFORMATION_MESSAGE);
    clearMemberFields(); // Call a method to clear the form fields
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
}
}

// Method to clear the form fields
private void clearMemberFields() {
    MemberFirstName.setText("");
    MemberLastName.setText("");
    MemberAdress.setText("");
    MemberPhone.setText("");
    MemberEmail.setText("");
}
```

5.3.3 Add Book Implementation

The screenshot shows a web application window titled "Library Management System". On the left is a vertical sidebar with buttons: Home, Add Book (highlighted), Add Member, Add Admin, Available Book, Barrow Book, Return Book, About, and Logout. The main area has a light green background and contains the following form fields:

- Title :
- Author :

First Name

Second Name
- Category :
- Publisher :
- PB Year :
- ISBN :
- Quantity :

At the bottom right of the form is a blue button labeled "Add Book".

```
private void addBook() {  
    String title = Title.getText().trim();  
    String firstName = FirstName.getText().trim();  
    String secondName = SecondName.getText().trim();  
    String category = (String) Categori.getSelectedItem();  
    String publisher = Publisher.getText().trim();  
    String publishYear = PublishYear.getText().trim();  
    String isbn = Isbn.getText().trim();  
    String quantityStr = quantity.getText().trim();  
  
    if (title.isEmpty() || firstName.isEmpty() || secondName.isEmpty() || category == null ||  
        publisher.isEmpty() || publishYear.isEmpty() || isbn.isEmpty() || quantityStr.isEmpty()) {  
        JOptionPane.showMessageDialog(this, "Please fill in all fields.", "Input Error",  
JOptionPane.ERROR_MESSAGE);  
        return;  
    }  
}
```

```

    }

    int quantity;
    try {
        quantity = Integer.parseInt(quantityStr);
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Quantity must be a number.", "Input Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    // SQL Queries
    String authorQuery = "INSERT INTO Authors (First_Name, Last_Name) VALUES (?, ?)";
    String bookQuery = "INSERT INTO Books (Title, Author_ID, Category_ID, Publisher,
Year_Published, ISBN, Quantity) VALUES (?, ?, ?, ?, ?, ?, ?)";

    try (Connection conn = DatabaseHelper.connect();
        PreparedStatement authorStmt = conn.prepareStatement(authorQuery,
PreparedStatement.RETURN_GENERATED_KEYS);
        PreparedStatement bookStmt = conn.prepareStatement(bookQuery)) {

        Integer authorId = getAuthorId(firstName, secondName, conn);

        // Insert author

        if (authorId == null) {
            // Author does not exist, insert new author
            authorStmt.setString(1, firstName);
            authorStmt.setString(2, secondName);
            authorStmt.executeUpdate();
            // Get the generated Author ID
            authorId = getAuthorId(firstName, secondName, conn); // Retrieve the new ID
        }

        // Get category ID
        int categoryId = getCategoryId(category, conn);

        if (categoryId == -1) {
            JOptionPane.showMessageDialog(this, "Category not found.", "Input Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }
    }

```

```

    }

    // Insert book
    bookStmt.setString(1, title);
    bookStmt.setInt(2, authorId);
    bookStmt.setInt(3, categoryId);
    bookStmt.setString(4, publisher);
    bookStmt.setString(5, publishYear);
    bookStmt.setString(6, isbn);
    bookStmt.setInt(7, quantity);
    bookStmt.executeUpdate();

    JOptionPane.showMessageDialog(this, "Book added successfully!", "Success",
JOptionPane.INFORMATION_MESSAGE);
    clearFields();

    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Database error: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
}

private int getCategoryId(String category, Connection conn) throws SQLException {
    String query = "SELECT Category_ID FROM Categories WHERE Category_Name = ?";
    try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString(1, category);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            return rs.getInt("Category_ID");
        }
    }
    return -1; // Return -1 if not found
}

private Integer getAuthorId(String firstName, String lastName, Connection conn) throws
SQLException {
    String query = "SELECT Author_ID FROM Authors WHERE First_Name = ? AND
Last_Name = ?";
    try (PreparedStatement stmt = conn.prepareStatement(query)) {
        stmt.setString(1, firstName);
        stmt.setString(2, lastName);
        ResultSet rs = stmt.executeQuery();

```

```

        if (rs.next()) {
            return rs.getInt("Author_ID"); // Return the existing author's ID

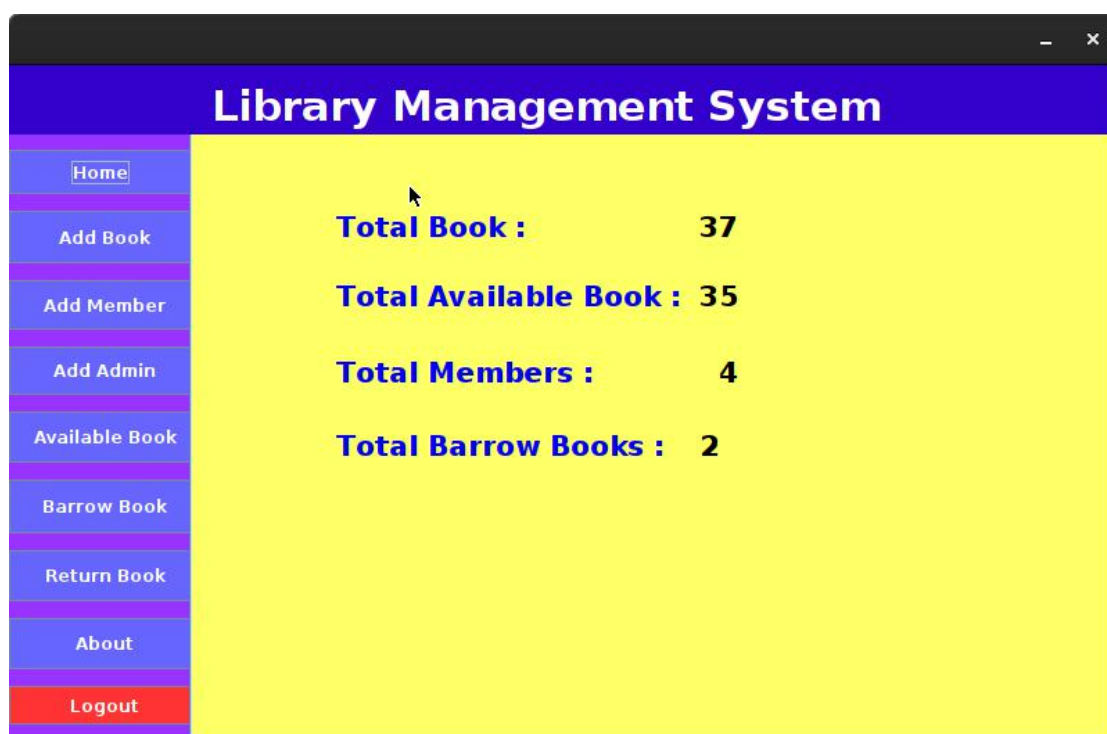
        }
    }
    return null; // Return null if the author does not exist

}

private void clearFields() {
    Title.setText("");
    FirstName.setText("");
    SecondName.setText("");
    Publisher.setText("");
    PublishYear.setText("");
    Isbn.setText("");
    Categori.setSelectedIndex(0);
    quantity.setText("");
}

```

5.3.4 Home Page Implementation



```

private void loadTotals() {

    Connection conn = null; // Declare conn outside the try block
    try {
        conn = DatabaseHelper.connect(); // Initialize conn
        conn.setAutoCommit(false); // Set auto-commit to false

        // Query to get total available books
        String totalAvailableBooksQuery = "SELECT SUM(Quantity) AS total FROM Books
WHERE Status = 'Available'";
        int totalAvailableBooks = 0; // Initialize available books count

        try (PreparedStatement stmt = conn.prepareStatement(totalAvailableBooksQuery);
            ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                totalAvailableBooks = rs.getInt("total");
                TotalavailableBook.setText(String.valueOf(totalAvailableBooks)); // Set only the
number
            }
        } catch (SQLException e) {
            JOptionPane.showMessageDialog(null, "Error retrieving available books count: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }

        // Total Members
        String totalMembersQuery = "SELECT COUNT(*) AS total FROM Members";
        try (PreparedStatement stmt = conn.prepareStatement(totalMembersQuery);
            ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                int totalMembers = rs.getInt("total");
                TotalMembers.setText(String.valueOf(totalMembers));
            }
        }

        // Count Total Borrowed Books (where Return_Date is NULL)
        String totalBorrowedBooksQuery = "SELECT COUNT(*) AS TotalBorrowedBooks
FROM Transactions WHERE Return_Date IS NULL";
        int totalBorrowedBooks = 0; // Initialize borrowed books count
    }
}

```

```

try (PreparedStatement stmt = conn.prepareStatement(totalBorrowedBooksQuery);
    ResultSet rs = stmt.executeQuery()) {
    if (rs.next()) {
        totalBorrowedBooks = rs.getInt("TotalBorrowedBooks");
        TotalBarrow.setText(String.valueOf(totalBorrowedBooks)); // Update the correct label
    }
} catch (SQLException e) {
    JOptionPane.showMessageDialog(null, "Error retrieving borrowed books count: " +
e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
}

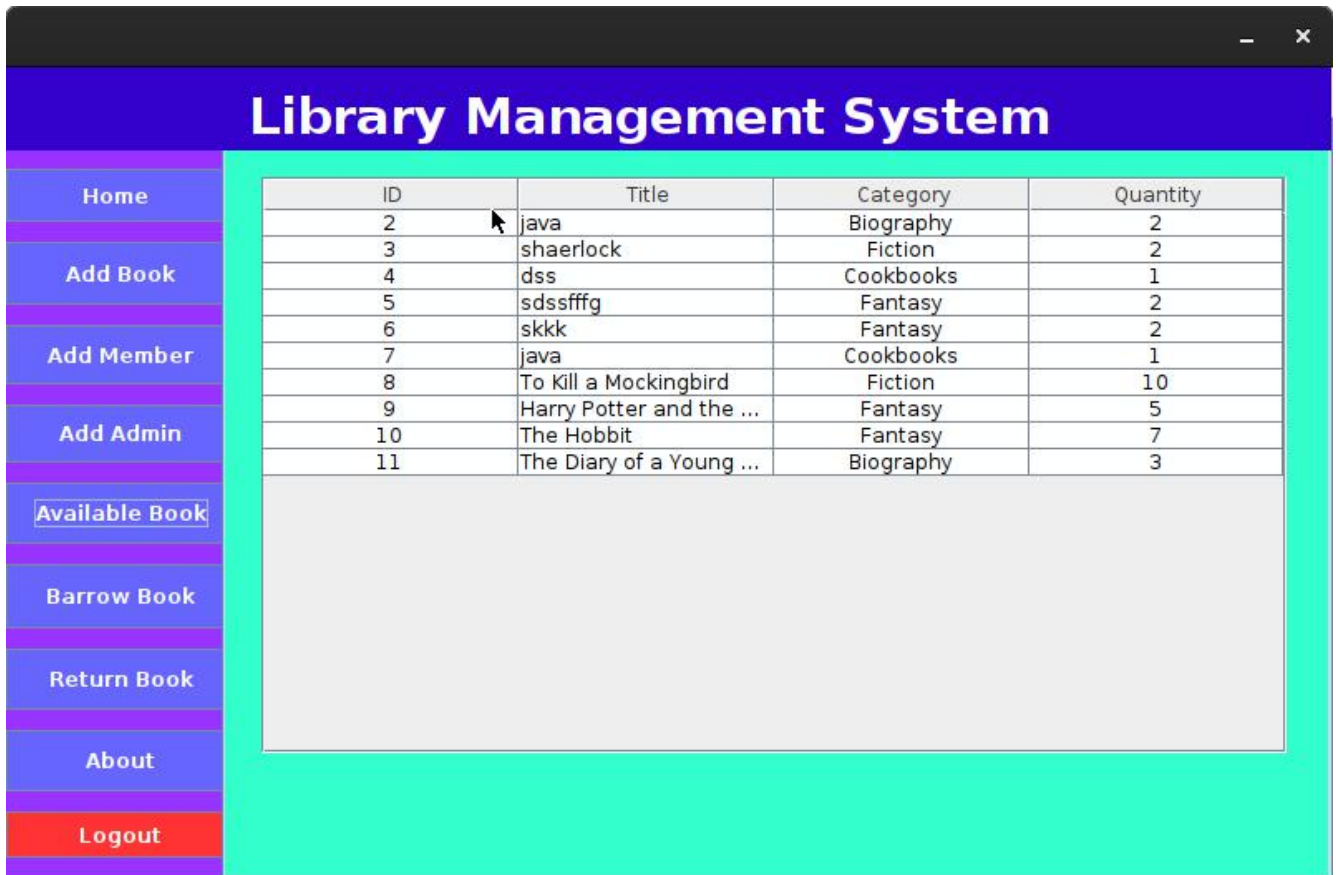
// Calculate total books
int totalBooks = totalAvailableBooks + totalBorrowedBooks;

// Update the JLabel with the total number of books
TotalBooks.setText(String.valueOf(totalBooks)); // Update the correct label

// Commit the transaction
conn.commit();
} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error loading totals: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    // Handle rollback in case of an error
    try {
        if (conn != null) {
            conn.rollback();
        }
    } catch (SQLException rollbackEx) {
        JOptionPane.showMessageDialog(this, "Error during rollback: " +
rollbackEx.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
    }
} finally {
    try {
        if (conn != null) {
            conn.setAutoCommit(true); // Reset auto-commit to true
            conn.close(); // Close the connection
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(this, "Error resetting auto-commit: " + e.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
    }
}
}

```


5.3.5 Available Books Table Implementation



```
private void loadAvailableBooks() {  
  
    String query = "SELECT b.Book_ID, b.Title, c.Category_Name, b.Quantity " +  
        "FROM Books b " +  
        "JOIN Categories c ON b.Category_ID = c.Category_ID " +  
        "WHERE b.Status = 'Available'";  
  
    try (Connection conn = DatabaseHelper.connect();  
        PreparedStatement stmt = conn.prepareStatement(query);  
        ResultSet rs = stmt.executeQuery()) {  
  
        DefaultTableModel model = (DefaultTableModel) BookTable.getModel();  
        model.setRowCount(0); // Clear existing rows  
  
        while (rs.next()) {  
            int bookId = rs.getInt("Book_ID");
```

```
String title = rs.getString("Title");
String categoryName = rs.getString("Category_Name");
int quantity = rs.getInt("Quantity");

model.addRow(new Object[]{bookId, title, categoryName, quantity});
}

BookTable.getColumnModel().getColumn(0).setCellRenderer(new CenterRenderer());
BookTable.getColumnModel().getColumn(1).setCellRenderer(new LeftRenderer());
BookTable.getColumnModel().getColumn(2).setCellRenderer(new CenterRenderer());
BookTable.getColumnModel().getColumn(3).setCellRenderer(new CenterRenderer());

} catch (SQLException e) {
    JOptionPane.showMessageDialog(this, "Error loading available books: " + e.getMessage(),
    "Database Error", JOptionPane.ERROR_MESSAGE);
}
}
```

6. APPENDIX

This appendix provides additional details regarding the user interfaces, coding practices, MySQL database design, and other relevant information related to the implementation of the Library Management System (LMS). The appendix is divided into multiple sections for clarity.

Appendix A: MySQL Database Design

A.1 Database Schema

This section outlines the database schema used in the LMS, including tables, fields, and relationships.

A.1.1 Users Table

```
MariaDB [LibraryManagementSys]> CREATE TABLE Members (  
  ->   Member_ID INT PRIMARY KEY AUTO_INCREMENT,  
  ->   First_Name VARCHAR(50) NOT NULL,  
  ->   Last_Name VARCHAR(50) NOT NULL,  
  ->   Address VARCHAR(255),  
  ->   Phone VARCHAR(15),  
  ->   Email VARCHAR(100) UNIQUE,  
  ->   Membership_Date DATE DEFAULT CURRENT_DATE  
  -> );  
Query OK, 0 rows affected (0.192 sec)  
MariaDB [LibraryManagementSys]> 
```

C.1.2 Books Table

```
sahan@debian: ~  
File Edit View Search Terminal Help  
-> );  
Query OK, 0 rows affected (0.155 sec)  
  
MariaDB [LibraryManagementSys]> CREATE TABLE Categories (  
->     Category_ID INT PRIMARY KEY AUTO_INCREMENT,  
->     Category_Name VARCHAR(50) UNIQUE NOT NULL  
-> );  
Query OK, 0 rows affected (0.195 sec)  
  
MariaDB [LibraryManagementSys]> CREATE TABLE Books (  
->     Book_ID INT PRIMARY KEY AUTO_INCREMENT,  
->     Title VARCHAR(255) NOT NULL,  
->     Author_ID INT,  
->     Category_ID INT,  
->     Publisher VARCHAR(100),  
->     Year_Published YEAR,  
->     ISBN VARCHAR(20) UNIQUE,  
->     Status ENUM('Available', 'Issued') DEFAULT 'Available',  
->     FOREIGN KEY (Author_ID) REFERENCES Authors(Author_ID),  
->     FOREIGN KEY (Category_ID) REFERENCES Categories(Category_ID)  
-> );  
Query OK, 0 rows affected (0.271 sec)  
  
MariaDB [LibraryManagementSys]> █
```

C.1.3 Borrowing Records Table

```
MariaDB [LibraryManagementSys]> CREATE TABLE Transactions (  
  ->   Transaction_ID INT PRIMARY KEY AUTO_INCREMENT,  
  ->   Book_ID INT,  
  ->   Member_ID INT,  
  ->   Issue_Date DATE DEFAULT CURRENT_DATE,  
  ->   Return_Date DATE,  
  ->   Fine DECIMAL(10, 2) DEFAULT 0.00,  
  ->   FOREIGN KEY (Book_ID) REFERENCES Books(Book_ID),  
  ->   FOREIGN KEY (Member_ID) REFERENCES Members(Member_ID)  
  -> );  
Query OK, 0 rows affected (0.231 sec)  
  
MariaDB [LibraryManagementSys]> 
```

7. REFERENCES

This section provides a list of supporting materials used in the development of the Library Management System (LMS). The references are organized according to the guidelines provided in Appendix B.

7.1 Books

1. Deitel, P. J., & Deitel, H. M. (2018). *Java: How to Program (11th Edition)*. Pearson Education.

This book provides comprehensive coverage of Java programming concepts and is a valuable resource for understanding object-oriented programming.

2. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems (7th Edition)*. Pearson Education.

This book covers the principles of database design and implementation, including SQL and relational database management systems.

7.2 Research Materials

1. Kumar, A., & Singh, R. (2020). "A Study on Library Management Systems: Challenges and Solutions." *International Journal of Library and Information Science*, 12(3), 45-52.

This research paper discusses the challenges faced by library management systems and proposes solutions for improving efficiency and user experience.

2. Smith, J. (2019). "The Role of Technology in Modern Libraries." *Library Technology Reports*, 55(4), 10-15.

This article explores how technology is transforming library services and the importance of implementing effective library management systems.

7.3 Web Links

Oracle. (n.d.). *Java SE Documentation*. Retrieved from <https://docs.oracle.com/javase/8/docs/>

This official documentation provides detailed information about Java SE, including APIs, tutorials, and best practices.

MySQL. (n.d.). *MySQL Documentation*. Retrieved from <https://dev.mysql.com/doc/>

This resource offers comprehensive documentation on MySQL, including installation guides, SQL syntax, and database management.

7.4 Other Relevant Information

1. Library Management System Project Guidelines. (2023). *University of XYZ*. Internal Document.

This document outlines the project requirements and guidelines for developing the Library Management System as part of the coursework.

2. Correspondence with Library Staff. (2023). Email communication regarding user requirements and system functionalities.

This correspondence provided valuable insights into the specific needs and challenges faced by the library, which informed the design and implementation of the LMS.

7.5 Online Forums and Communities

1. Stack Overflow. (n.d.). *Java and MySQL Questions*. Retrieved from <https://stackoverflow.com/questions/tagged/java+mysql>

This online community is a valuable resource for troubleshooting and finding solutions to common programming issues related to Java and MySQL.