King Abdulaziz University
Faculty of Computing & Information Technology
Department of Computer Science
CPCS371 | Networks Project, Spring Semester 2024
Prepared for: Dr. Ohood Alzamzami

A Client-Server Application
Library Management System (LMS)

Student Name	Student ID	Section
Joud Jamal Alkishi	20*****	EAR
Sara Abdulaziz Alzahrani	21*****	VAR
Rayyanah Abdulkarim Alshehri	21*****	VAR
Shahad Abdulaziz Salman	21*****	VAR

Submit Date: 12/5/2024

Table of Contents

List of Figures/Tables

1. Introduction

1.1 Project Background

1.1.1 Client-server application paradigm

A client-server architecture allocates work to two sorts of software components: clients and servers. Client-side programs operate on end-user devices and deliver requests to server-provided services. The server side receives these requests, processes them, and returns suitable answers with the required information or services. This communication follows a request-response pattern and employs communication protocols, which are rules that control how information is transmitted between parties. Popular protocols include TCP (Transmission Control Protocol), DNS (Domain Name System), and IP (Internet Protocol). This design supports distributed computing by making it possible for programs to communicate and exchange resources effectively across networks **[1]**.

1.1.2 What is LMS?

Our Library Management System (LMS) is based on the client-server approach. Users interact with the system via a client application, which communicates with a central server. The server runs the LMS logic and stores databases of registered users and available books. Librarians have administrative control over the book inventory, while members can browse, borrow, and return books.

Librarians can add, and delete books by entering their titles and quantities. Members may browse books, borrow, and return them. Borrowing decreases available copies, and returning refreshes the inventory. The system tracks borrowing and return dates, imposing a 24-hour borrowing restriction per book.

Members who log in receive information about book availability and return dates. This architecture promotes efficient library resource management and includes features that enable librarians and members to engage successfully with the LMS.

1.1.3 Network protocols used in the development of the project and their purpose of use.

TCP: The purpose of the protocol's objective is to transmit a stream of bytes between applications operating on devices linked by an IP network in a reliable, ordered, and error-checked manner. TCP assures that data transferred from one application to another is precisely and sequentially delivered, with no mistakes or packet loss **[2]**.

1.2 Objectives

- **Efficient Resource Management:** Create a powerful LMS employing a client-server approach to support simplified administration of library resources, including tracking of book inventory and member transactions.
- **Secure Authentication and Access Control:** Implement secure login techniques to authenticate users (librarians and members), with access control based on roles and permissions within the LMS.
- **Real-time Information Updates:** Enable real-time updates for book availability and borrowing statuses, so that members and librarians receive timely information when they log in or interact with the system.
- **Reliable Communication Protocols:** Implement the TCP protocol to enable consistent and error-free communication between clients and the central server, hence maintaining data integrity and order.

2. System Architecture and Design

2.1 Overall Architecture

Our Library Management System (LMS) architecture comprises three main components: the client-side application, the central server, and the database.

- **Client-Side Application**

The client-side application provides a user-friendly interface for members and librarians to interact with the LMS. Users access functionalities such as browsing, borrowing, and returning books through this application. Additionally, the client handles user authentication, sending login credentials securely to the server for validation. Upon successful authentication, the client receives real-time updates on book availability and return dates.

- **Central Server**

The central server serves as the backbone of the LMS, managing all data and business logic. It receives requests from client applications, processes them, and responds accordingly. The server stores databases of registered users and available books, ensuring efficient resource management. Furthermore, it implements secure authentication mechanisms to authenticate users and enforce access control based on user roles and permissions.

- **Database**

The database component stores persistent data related to users, books, borrowing transactions, and system configurations. It serves as the primary data repository for the LMS, ensuring data integrity, reliability, and scalability. The server interacts with the database to retrieve and update information as needed, ensuring that the LMS operates smoothly and efficiently.

- **Communication Protocols**

The client-server communication is facilitated by the TCP protocol, ensuring reliable and ordered transmission of data between the client and the server. This protocol guarantees that information is delivered accurately and without loss, maintaining data integrity and order throughout the interaction.

2.2 Interaction Diagrams

Librarian-Client

Figure 1: Interaction diagram for a librarian

Member-Client

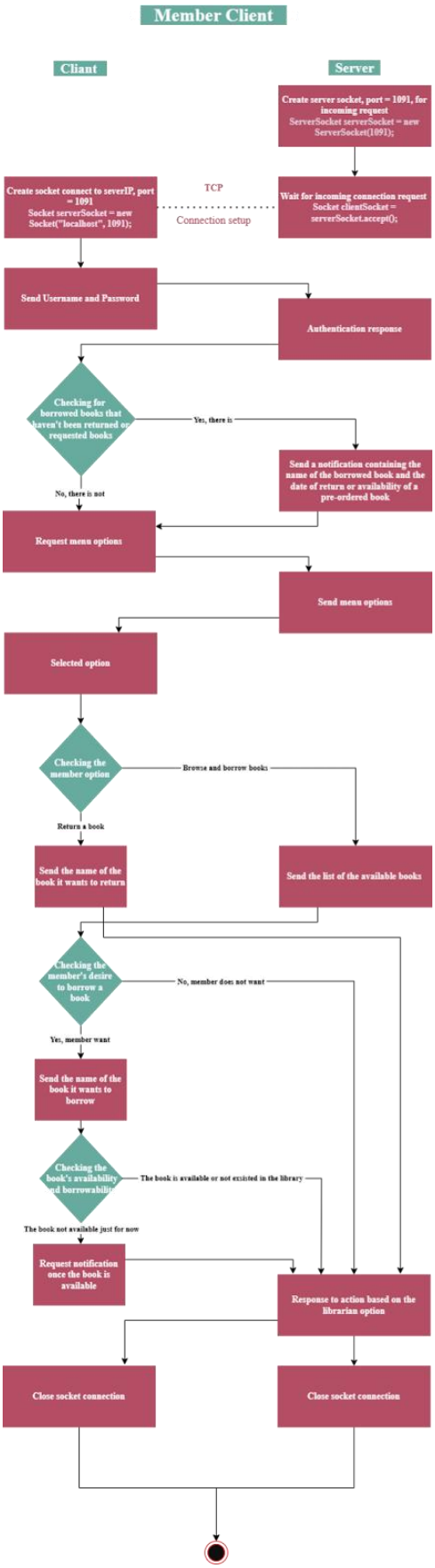


Figure 2: Interaction diagram for a member [3]

3. Project Implementation

3.1 Overview of the libraries, and main functions of client-server app

Libraries:

- **Client Side:**

1. Java Networking (java.net):
   - Socket: Establishes a connection to the server.
2. Input/Output (java.io):
   - BufferedReader, PrintWriter: Used for reading from and writing to the socket streams.
3. Utility (java.util.Scanner):
   - Scanner: Reads user input from the console.

- **Server Side:**

1. Java Networking (java.net):
   - ServerSocket: Used to create a server socket and listen for client connections.
   - Socket: Represents a client-server communication endpoint.

2. Input/Output (java.io):
   - BufferedReader: Reads text from a character-input stream, buffering characters to provide for the efficient reading of characters, arrays, and lines.
   - PrintWriter: Prints formatted representations of objects to a text-output stream.
   - File, FileReader, FileWriter, BufferedWriter: Used for file I/O operations to read/write users and books data from/to text files.

3. Collections (java.util):
   - Map: Used to store user credentials (users) and book information (books).
   - ArrayList: Used to store lists of requested, borrowed, and returned books.

4. Threading:
   - Thread: Handles each client connection separately by creating a new thread (ClientHandler) for each client.

5. Date and Time (java.util.Calendar, java.util.Date, java.text.SimpleDateFormat):
   - Used for managing book return dates and formatting them appropriately.

Main Functions:

- **Client Side:**
  - main: Establishes a connection to the server, reads user input, and communicates with the server based on the user's role (librarian or member).
- **Server Side:**
  - main: Starts the server, reads user and book data from files, and listens for incoming connections.
  - ClientHandler: Inner class to handle each client connection separately.

3.2 Implementation Details

- Server Side:

`AddBook(PrintWriter out, BufferedReader in, String filePath, String title, int copies)`

The AddBook is a librarian method that helps librarians in the Library Management System (LMS) to add new books easily. It takes the book's title and number of copies they want to add. It uses basic tools in Java to update the library's records right away. Librarians get simple messages to know if the book was added successfully or if it already exists. And it's designed to handle any unexpected problems that might happen when saving the changes.

```
// AddBook: a librarian method, to add new book to the database.
private void AddBook(PrintWriter out, BufferedReader in, String filePath, String title, int copies) throws IOException {
            BufferedWriter bw = new BufferedWriter(new FileWriter(filePath, true));
            if (!books.containsKey(title)) {
                bw.write(title + " " + copies);
                bw.newLine();
                bw.flush();
                books.put(title, copies);
                out.println("✔ Book added successfully ✔");
            } else {
                out.println("Book already exists.");
            }
        }
```

Figure 3: Implementation of AddBook method

UpdateBook(PrintWriter out, BufferedReader in)

The UpdateBook is a librarian method in the Library Management System (LMS). It helps the librarian to change the number of copies of a book. It takes input from the librarian with the book title they want to update. If the book is in the library, they're asked to enter a new number of copies. Then, the method updates the book's information and saves the changes. If the book isn't found, the librarian is told so. This method makes it easy for librarians to keep track of how many copies of each book the library has.

```
// UpdateBook: a librarian method, to update book info > num of copies.
private void UpdateBook(PrintWriter out, BufferedReader in) throws IOException {
            out.println("➤ Enter book title to update:");
            String title = in.readLine();

            if (books.containsKey(title)) {
                out.println("➤ Enter new number of copies:");
                int newCopies = Integer.parseInt(in.readLine());

                books.put(title, newCopies);

                rewriteBooksToFile("books.txt");

                out.println("✔ Book updated successfully ✔");
            } else {
                out.println("Book not found.");
            }
        }
```

Figure 4: : Implementation of UpdateBook method

DeleteBook(PrintWriter out, BufferedReader in)

The DeleteBook is a librarian method in the Library Management System (LMS) to delete a specific book from the library's collection. It askes the librarian to enter the book title they want to delete. If the book is found in the library, it's deleted from the system's records, and the changes are saved. The librarian receives a message confirming the successful deletion. If the book isn't found, the librarian is told so. This method helps librarians manage the library's inventory by allowing them to remove unwanted or outdated books easily.

```
// DeleteBook: a librarian method, to delete a specific book.
private void DeleteBook(PrintWriter out, BufferedReader in) throws IOException {
            out.println("➤ Enter book title to delete:");
            String title = in.readLine();

            if (books.containsKey(title)) {
                // Remove the book from the in-memory map
                books.remove(title);

                // Update the books.txt file to reflect the deletion
                rewriteBooksToFile("books.txt");

                out.println("✔ Book deleted successfully ✔");
            } else {
                out.println("Book not found.");
            }
        }
```

Figure 5: : Implementation of DeleteBook method

`BrowseBook(PrintWriter out, BufferedReader in)`

The BrowseBook is a member method in the Library Management System (LMS). It offers members a comprehensive view of available books within the library. When invoked, it scans the library's inventory, counting and displaying the number of available books. Also, it presents detailed information about each available book, including its title and the number of copies available. This user-friendly presentation enables members to browse through the library's collection efficiently and make informed decisions about book selections for borrowing.

```
// BrowseBook: a member method, to display a list of available books and browse
them.
private void BrowseBook(PrintWriter out, BufferedReader in) throws IOException {
            int count = 0;
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getValue() != 0) {
                    count++;
                }
            }

            out.println(count);  // Send the number of books to the client
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getValue() != 0) {
                    out.println("• Title: " + entry.getKey() + " - Copies: " +
                    entry.getValue());
                }
            }
        }
```

Figure 6: : Implementation of BrowseBook method

`requestedBooks(PrintWriter out, BufferedReader in, ArrayList<String> returnedBooks, String username)`

The requestedBooks is a method in the Library Management System (LMS) that provides personalized notifications to members regarding the availability of previously requested books. It distinguishes between users based on their usernames, then examines the list of returned books to identify matches with requested titles for each user. Upon finding available books, it notifies users about the specific titles now accessible and updates them on the count of available titles. The method ensures accurate notifications by removing notified books from the user's list of requested titles, thereby streamlining communication and enhancing member satisfaction with timely access to desired resources.

```java
// requestedBooks: a member method, to send notification for availability of requested books.
private void requestedBooks(PrintWriter out, BufferedReader in, ArrayList<String> returnedBooks, String username) throws IOException {
            ArrayList<String> availableBooks1 = new ArrayList<>();
            ArrayList<String> availableBooks2 = new ArrayList<>();
            int countAvailable1 = 0; // Correctly count the number of borrowed books
            int countAvailable2 = 0;

            if (username.equals("Sherlock")) {
                for (String book : returnedBooks) {
                    if (requestedBooks1.contains(book)) {
                        availableBooks1.add(book);
                        countAvailable1++;
                    }
                }
                // Send the count of available books to the client
                out.println(countAvailable1);

                // Notify the user about the availability of requested books
                if (countAvailable1 > 0) {
                    out.println("🔔 Notification ▸ The following requested books are now available:");
                    for (String availableBook : availableBooks1) {
                        out.println(availableBook);
                    }
                    // Clear the list of requested books since they have been notified
                    requestedBooks1.removeAll(availableBooks1);
                }
            } else {
                for (String book : returnedBooks) {
                    if (requestedBooks2.contains(book)) {
                        availableBooks2.add(book);
                        countAvailable2++;
                    }
                }
                // Send the count of available books to the client
                out.println(countAvailable2);

                // Notify the user about the availability of requested books
                if (countAvailable2 > 0) {
                    out.println("🔔 Notification ▸ The following requested books are now available:");
                    for (String availableBook : availableBooks2) {
                        out.println(availableBook);
                    }
                    // Clear the list of requested books since they have been notified
                    requestedBooks2.removeAll(availableBooks2);
                }

            }
        }
```

Figure 7: : Implementation of requestedBooks method

`BorrowBook(PrintWriter out, BufferedReader in, String bookToBorrow, String username)`

The BorrowBook is a member method that allows library members to borrow books from the library. It checks if the requested book is available in stock. If the book is available, it decrements the number of copies and notifies the user about the successful borrowing. If the book is not available, it prompts the user if they want to be notified when the book is back in stock. If requested, the user will receive a notification upon the book's availability from requestedBooks method (figure 7). The method ensures accurate book borrowing and provides timely notifications for out-of-stock items, enhancing user experience and library management efficiency.

```
// BorrowBook: a member method, to borrow a book and to be notified if the book is not instock.
private void BorrowBook(PrintWriter out, BufferedReader in, String bookToBorrow, String username) throws
IOException {
            boolean bookFound = false;
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getKey().equalsIgnoreCase(bookToBorrow)) {
                    bookFound = true;
                    int copies = entry.getValue();
                    if (username.equals("Sherlock")) {
                        if (copies == 0) {
                            out.println("Book is not available ✘");
                            out.println("➤ Would you like to be notified when the book is in stock?");
                            String notifyUser = in.readLine();
                            if (notifyUser.equalsIgnoreCase("yes")) {
                                out.println("✔ You will be notified when the book is in stock ✔");
                                requestedBooks1.add(bookToBorrow);

                            }
                        } else {
                            books.put(entry.getKey(), copies - 1); // Update the number of copies
                            out.println("➤ You borrowed : " + entry.getKey());
                            out.println("🛜 THE BORROWED BOOK SHOULD BE RETURNED WITHIN 24 HOURS 🛜");
                            BooksBorrowed1.add(bookToBorrow);
                            break;
                        }
                    } else {
                        if (copies == 0) {
                            out.println("Book is not available ✘");
                            out.println("➤ Would you like to be notified when the book is in stock?");
                            String notifyUser = in.readLine();
                            if (notifyUser.equalsIgnoreCase("yes")) {
                                out.println("✔ You will be notified when the book is in stock ✔");
                                requestedBooks2.add(bookToBorrow);

                            }
                        } else {
                            books.put(entry.getKey(), copies - 1); // Update the number of copies
                            out.println("➤ You borrowed : " + entry.getKey());
                            out.println("🛜 THE BORROWED BOOK SHOULD BE RETURNED WITHIN 24 HOURS 🛜");
                            BooksBorrowed2.add(bookToBorrow);
                            break;
                        }

                    }
                    break; // Exit the loop once the book is found and processed
                }
            }
            rewriteBooksToFile("books.txt");
            if (!bookFound) {
                out.println("Book not found in the library ✘");
            }
        }
```

Figure 8: : Implementation of BorrowBook method

`ReturnBook(PrintWriter out, BufferedReader in, String bookToReturn, String username)`

The ReturnBook is a member method that facilitates the return of borrowed books. It checks if the returned book exists in the library's collection. If the book is found, it updates the number of copies accordingly and confirms the successful return to the user. Additionally, it removes the returned book from the borrower's list and adds it to the list of returned books. The method ensures accurate book returns and maintains proper inventory records, contributing to effective library management.

```
// ReturnBook: a member method, to return books.
private void ReturnBook(PrintWriter out, BufferedReader in, String bookToReturn, String username) throws IOException {
            boolean bookFound = false;
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getKey().equalsIgnoreCase(bookToReturn)) {
                    bookFound = true;
                    int copies = entry.getValue();
                    books.put(entry.getKey(), copies + 1); // Correctly update the number of copies
                    out.println("➤ You Returned : " + entry.getKey());
                    if (username.equals("Sherlock")) {
                        BooksBorrowed1.remove(bookToReturn);
                        returnedBooks.add(bookToReturn);
                    } else {
                        BooksBorrowed2.remove(bookToReturn);
                        returnedBooks.add(bookToReturn);
                    }
                    break;
                }
            }
            rewriteBooksToFile("books.txt"); // Move file writing outside the loop to avoid redundancy
            if (!bookFound) {
                out.println("Book not found in the library ✘");
            }
        }
```

Figure 9: : Implementation of ReturnBook method

`sendNotifications(PrintWriter out, String username)`

The sendNotifications is a method responsible for informing users about the return dates of books they have borrowed. For each user, it retrieves the list of borrowed books and their corresponding return dates. Then, it sends this information to the user. If a user has no borrowed books, no notification is sent.

```
// sendNotifications: a member method, to send return date notification for user borrowed books.
private void sendNotifications(PrintWriter out, String username) {

            if (username.equals("Sherlock")) {
                int counter1 = BooksBorrowed1.size(); // Count the number of borrowed books
                out.println(counter1); // Send the number of borrowed books to the client

                // Notification for borrowed books return dates
                if (!BooksBorrowed1.isEmpty()) {
                    out.println("🔔 Notification ➤ Your borrowed books return dates:");
                    for (String borrowedBook : BooksBorrowed1) {
                        String borrowedReturnDate = getReturnDateForBook(borrowedBook); // Use Date type
                        out.println(borrowedBook + " - Return Date: " + borrowedReturnDate);
                    }
                }
            } else {
                int counter2 = BooksBorrowed2.size(); // Count the number of borrowed books
                out.println(counter2); // Send the number of borrowed books to the client
                if (!BooksBorrowed2.isEmpty()) {
                    out.println("🔔 Notification ➤ Your borrowed books return dates:");
                    for (String borrowedBook : BooksBorrowed2) {
                        String borrowedReturnDate = getReturnDateForBook(borrowedBook); // Use Date type
                        out.println(borrowedBook + " - Return Date: " + borrowedReturnDate);
                    }
                }

            }
        }
```

Figure 10: : Implementation of sendNotifications method

4. Test Cases and Results

1. Member Browse book

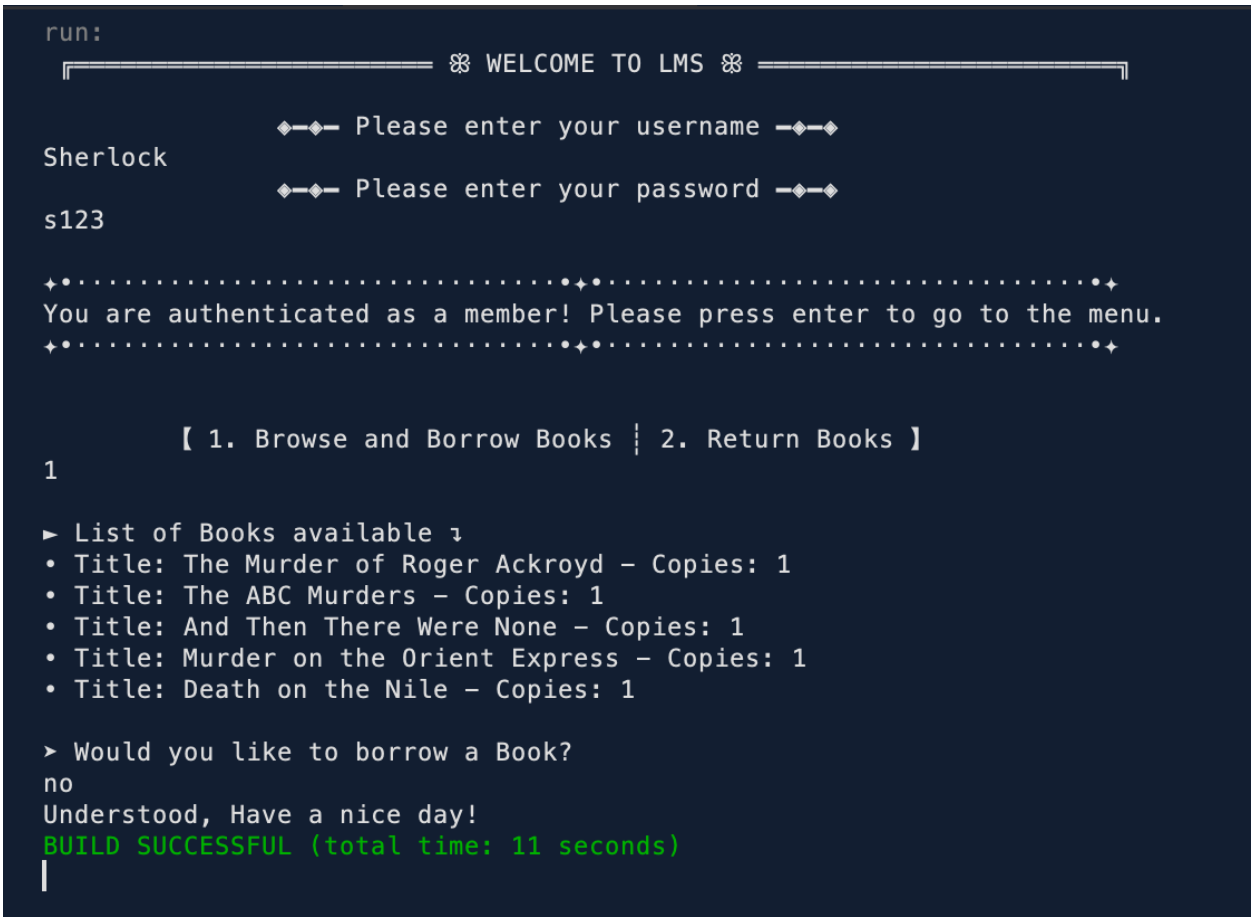


Figure 11: Test case & Result of Member Browse Book

In this scenario, a logged-in member browses the available books in the Library Management System (LMS). They view a list of books with details like titles, and availability. The system offers the option to borrow a book, but the member chooses not to borrow at that time. The member successfully completes their browsing session without initiating any book borrowing. This illustrates the member's ability to explore book options and make decisions within the LMS interface.

2. Member Borrow book

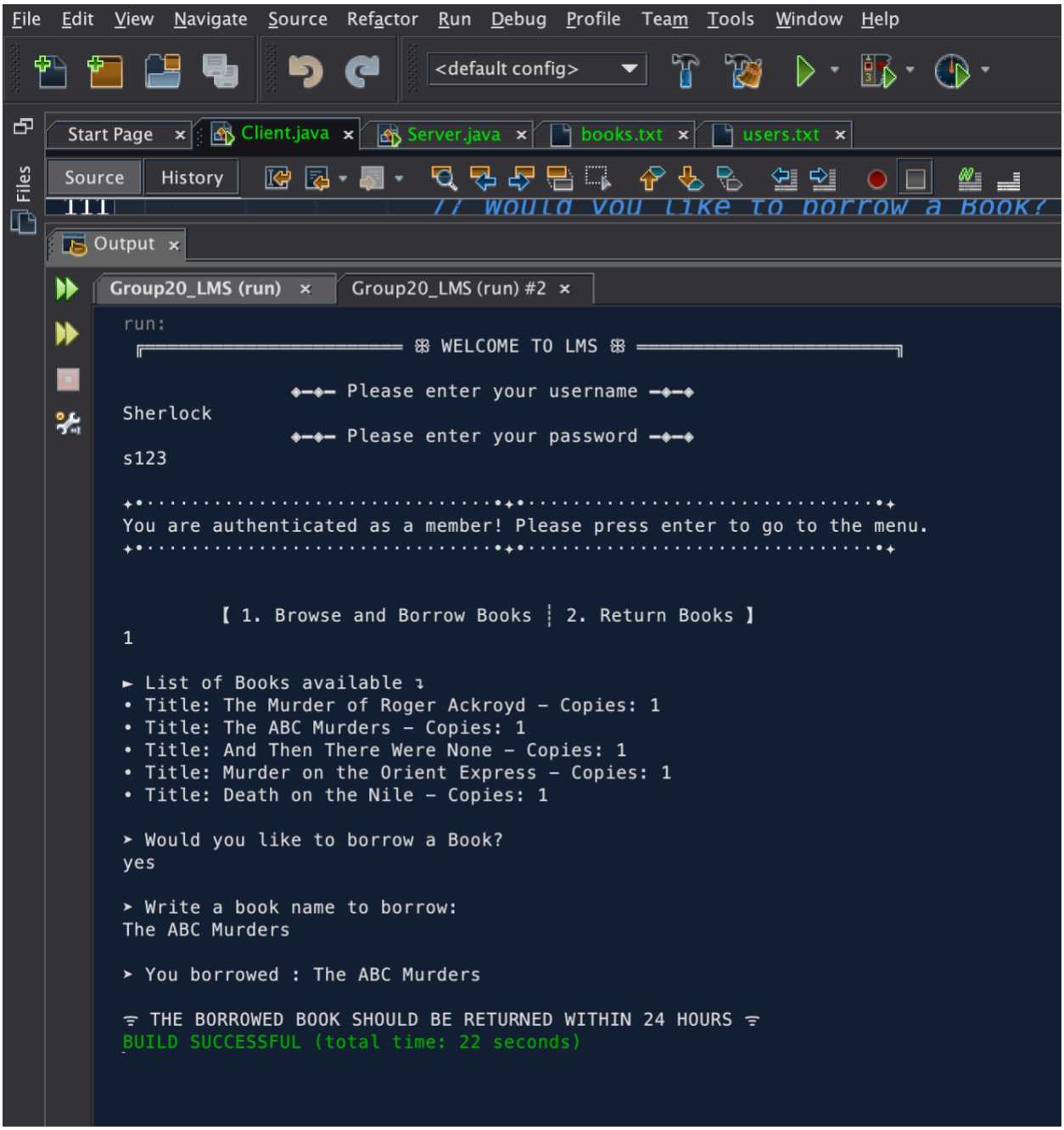


Figure 12: Test case & Result of Member Borrow Book

In this scenario, a logged-in member navigates to the Library Management System (LMS) to borrow a specific book, "The ABC Murders." After selecting the book for borrowing, the system confirms the transaction, and the member now has 24 hours until the return deadline. This example showcases the member's ability to borrow a book through the LMS and highlights the accompanying borrowing period, providing a clear timeline for returning the borrowed item.

3. Member Return book

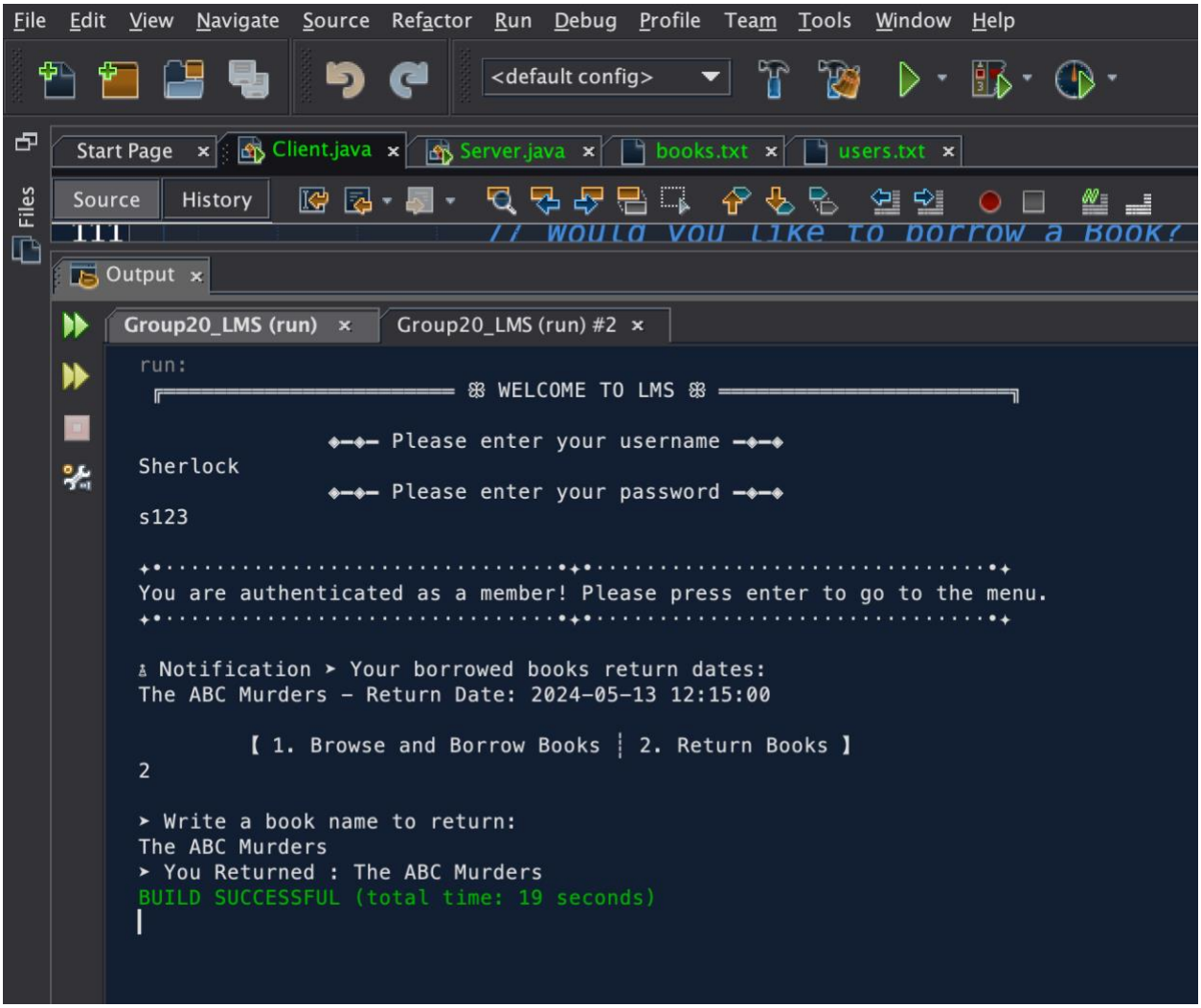


Figure 13: Test case & Result of Member Return Book

In this scenario, a member who has previously borrowed a book logs into the Library Management System (LMS). Upon logging in, a notification appears indicating the book they have borrowed, titled "The ABC Murders," and the return deadline, which is within 24 hours. The member is then presented with a menu offering options to borrow another book or to return the currently borrowed book. Choosing to return the book, the member initiates the return process successfully through the LMS interface. This scenario demonstrates the LMS's capability to notify members of their borrowed items upon login and allows them to conveniently manage their borrowed books by returning them within the specified timeframe.

4. Librarian Add book

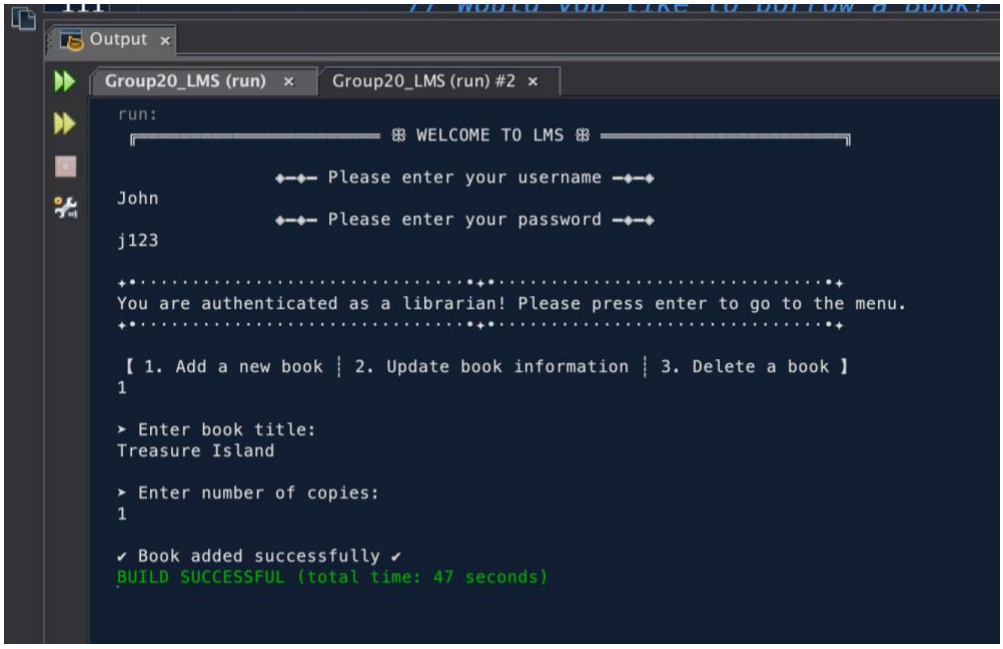


Figure 14: Test case & Result of Librarian Add Book

In this scenario, a librarian logs into the Library Management System (LMS) and adds a new book titled "Treasure Island" with one copy using the librarian-specific menu. Upon completion, a confirmation message appears, indicating the successful addition of the book to the system. This demonstrates the LMS's efficiency in allowing librarians to manage the library's collection by seamlessly adding new books and maintaining accurate inventory records. The clear feedback message enhances the user experience by providing immediate confirmation of the completed task within the LMS interface.

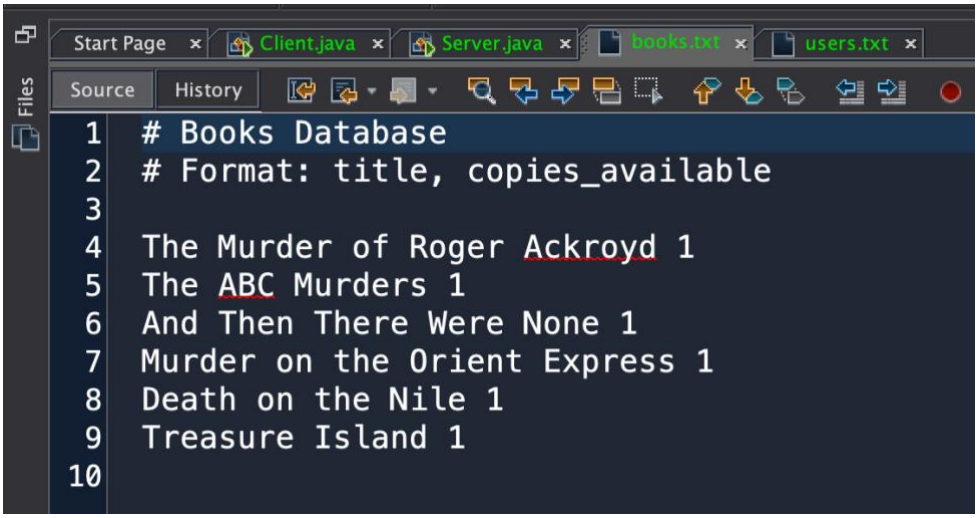


Figure 15: Book added to database

Upon completion of the addition process, it is evident that "Treasure Island" has been successfully incorporated into the database.

5. Librarian Update book info

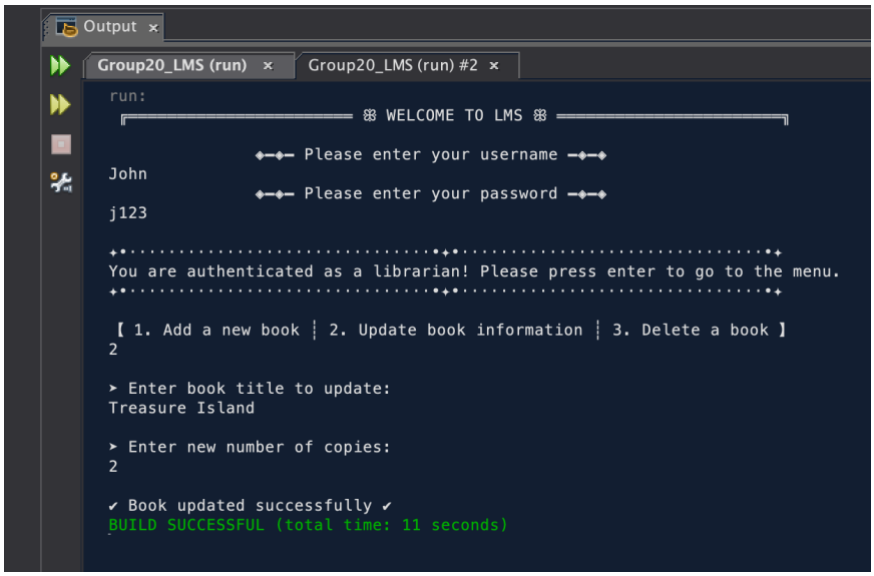


Figure 16: Test case & Result of Librarian Update Book Info

In this scenario, a librarian logs into the Library Management System (LMS) with the intention of updating the information for the book "Treasure Island." Upon logging in, the librarian navigates to the menu and selects the option to update the book's information. Specifically, the librarian modifies the number of copies for "Treasure Island" from 1 to 2. After successfully updating the book's information, the LMS confirms the changes with a message, affirming that the details for "Treasure Island" have been successfully updated in the database. This scenario exemplifies the LMS's functionality in enabling librarians to efficiently manage the library's inventory by making necessary updates to book information.

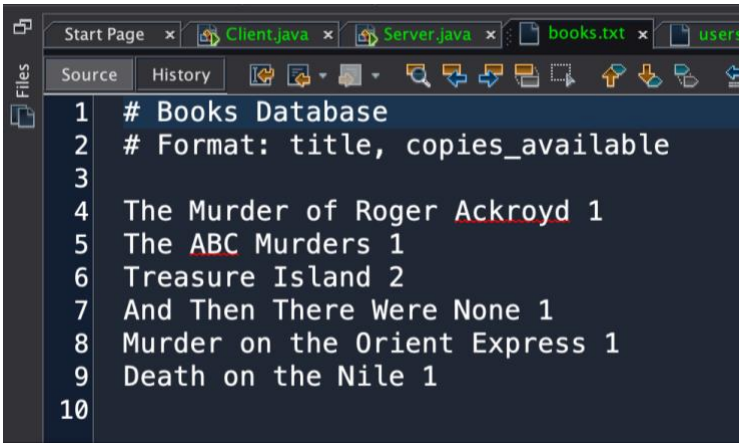


Figure 17: Book info updated on the database

Upon completion of the updating process, it is evident that "Treasure Island" new copies has been successfully incorporated into the database

6. Librarian Delete book

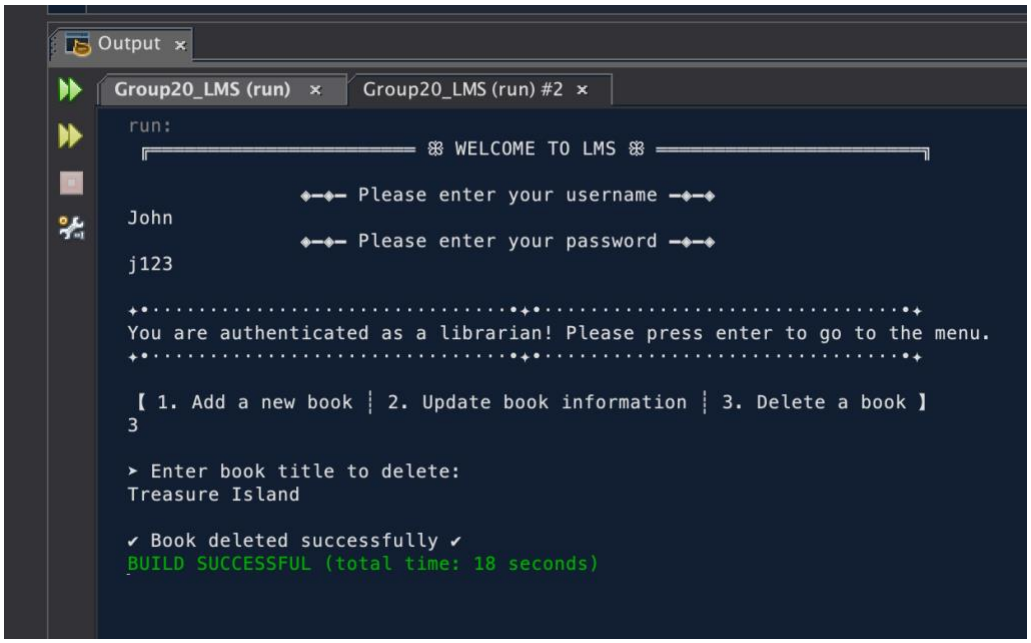


Figure 18: : Test case & Result of Librarian Delete Book

In this scenario, a librarian accesses the Library Management System (LMS) and navigates to the librarian-specific menu. From the menu options, the librarian selects the "Delete Book" feature. Subsequently, the librarian is prompted to enter the name of the book to be deleted, which in this case is "Treasure Island." After entering the book's name and confirming the deletion request, a notification promptly appears on the LMS interface, confirming that "Treasure Island" has been successfully deleted from the system database. This demonstrates the LMS's capability to facilitate efficient book management for librarians, providing clear and immediate feedback upon completion of deletion tasks within the system.

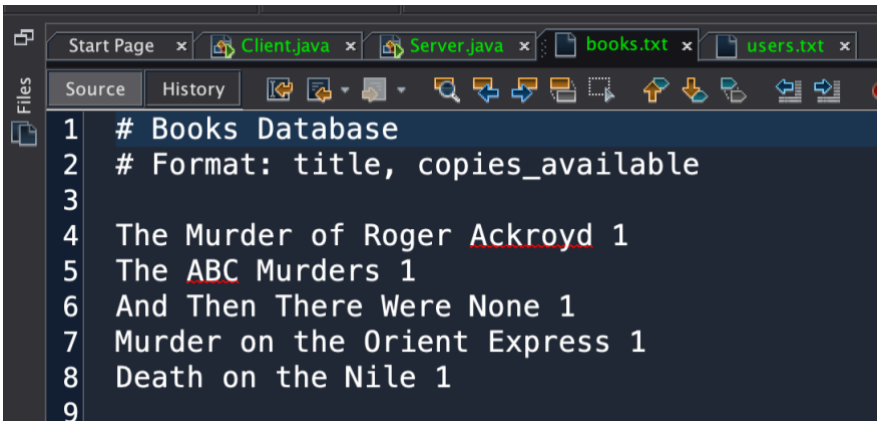


Figure 19: Book is deleted from the database

Upon completion of the deleting process, it is evident that "Treasure Island" has been successfully removed from the database.

7. Return date notification

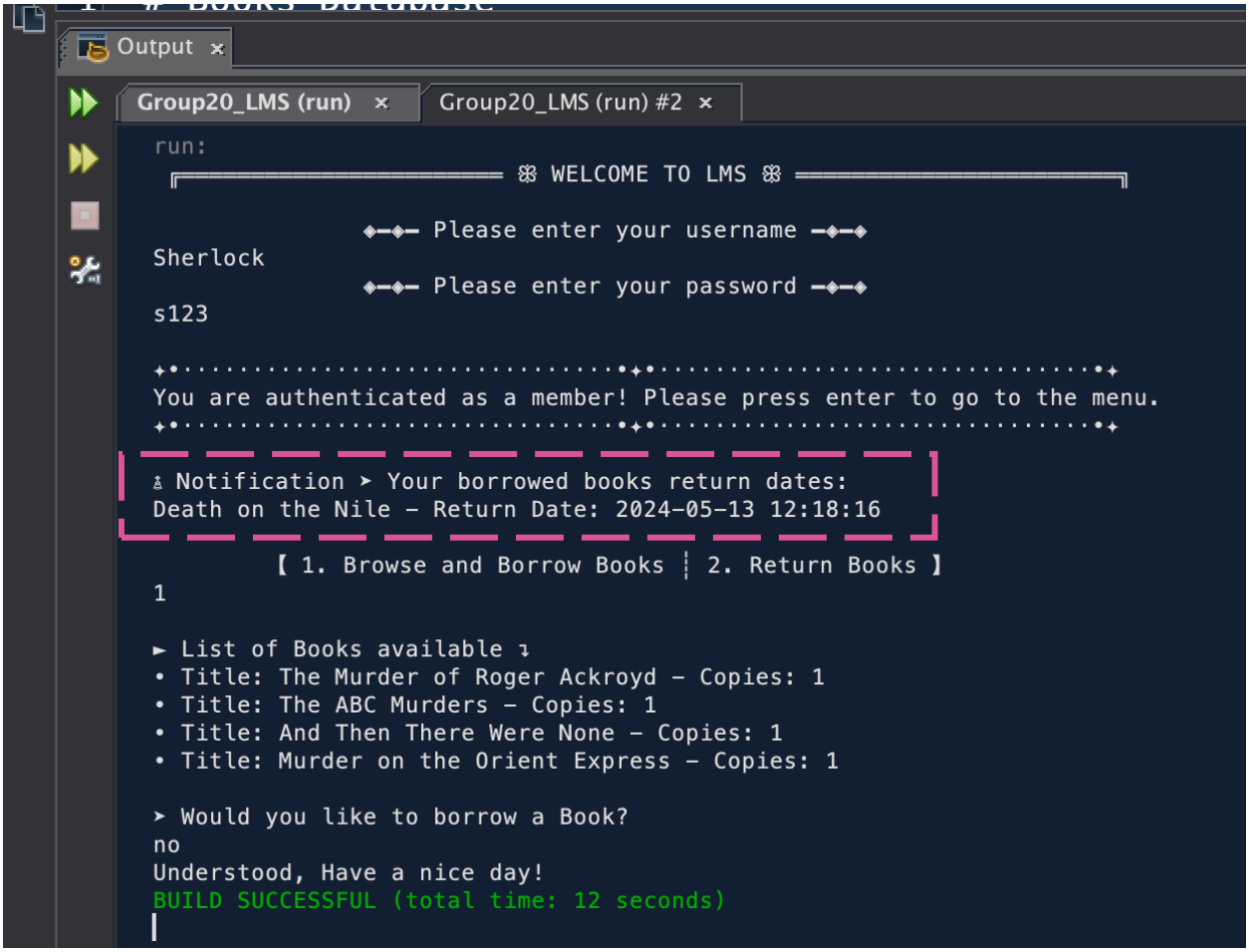


Figure 20: Borrowed book return date notification

In this scenario, a member who has previously borrowed a book logs into the Library Management System (LMS) and is presented with a notification displaying the details of the borrowed book, "Death on the Nile," along with the return deadline date. Subsequently, the member is presented with a menu offering options to browse and borrow a book or to return the currently borrowed book. This scenario illustrates the LMS's capability to notify members of their borrowed items and provide a user-friendly interface for managing book transactions and browsing within the library system.

8. Book availability notification

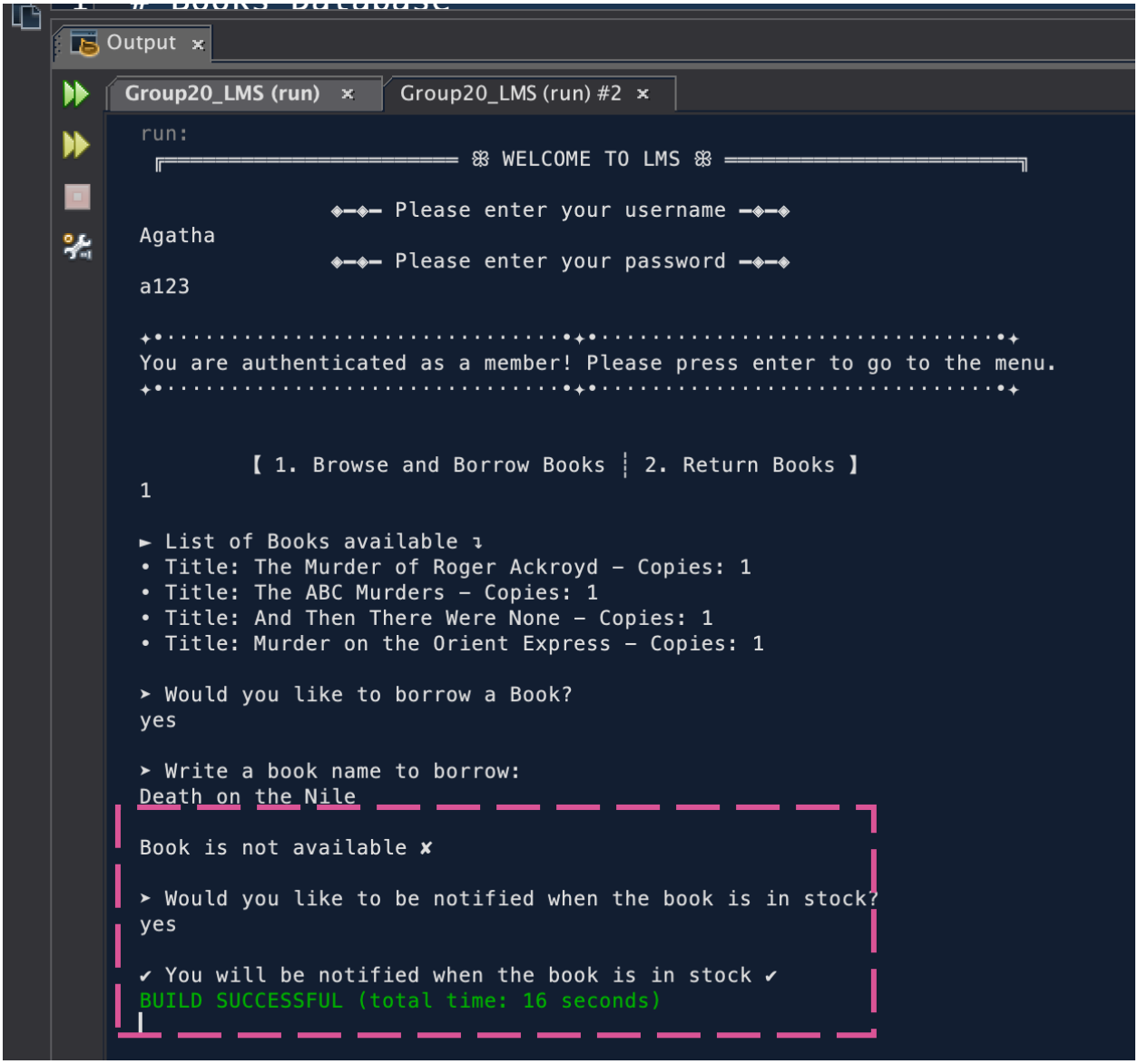


Figure 21: Client chose to be notified when the book is available

In this scenario, a logged-in user who is authenticated as a member attempts to borrow a book that is currently unavailable in the Library Management System (LMS). Upon attempting to borrow the book, the user receives a notification indicating that the requested book is out of stock. The system then prompts the user with a message asking if they would like to be notified when the book becomes available. The user responds affirmatively by selecting "yes" to receive a notification when the book is back in stock. This interaction demonstrates the LMS's capability to engage users proactively by offering to notify them when requested items become available, enhancing the user experience and facilitating efficient access to desired library resources

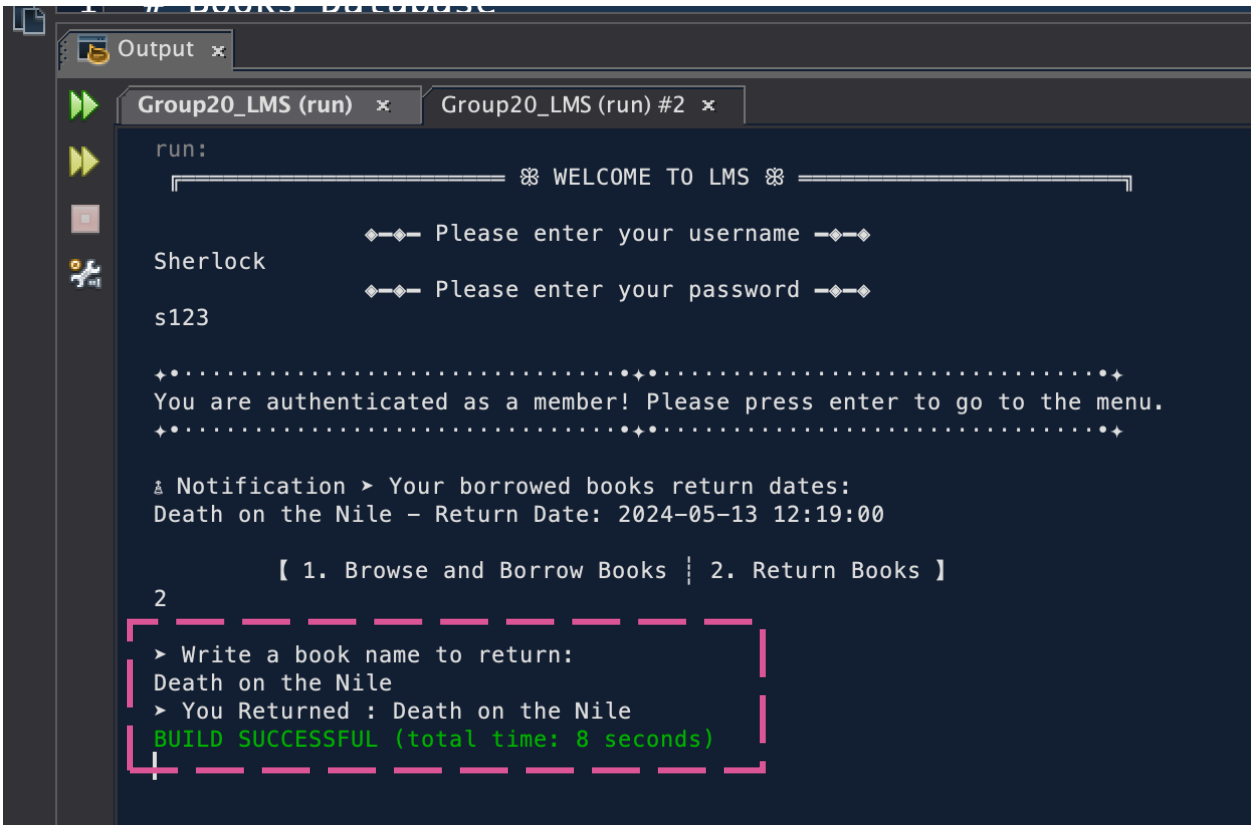


Figure 22: The requested book, is now returned so it's available

A logged-in user who is authenticated as a member has previously borrowed a book titled "Death on the Nile." The user decides to return the book by accessing the Library Management System (LMS) and selecting the option to return a book. The user enters the title "Death on the Nile" to specify the book for return. Upon submitting the return request, a confirmation message promptly appears, notifying the user that the book "Death on the Nile" has been successfully returned.

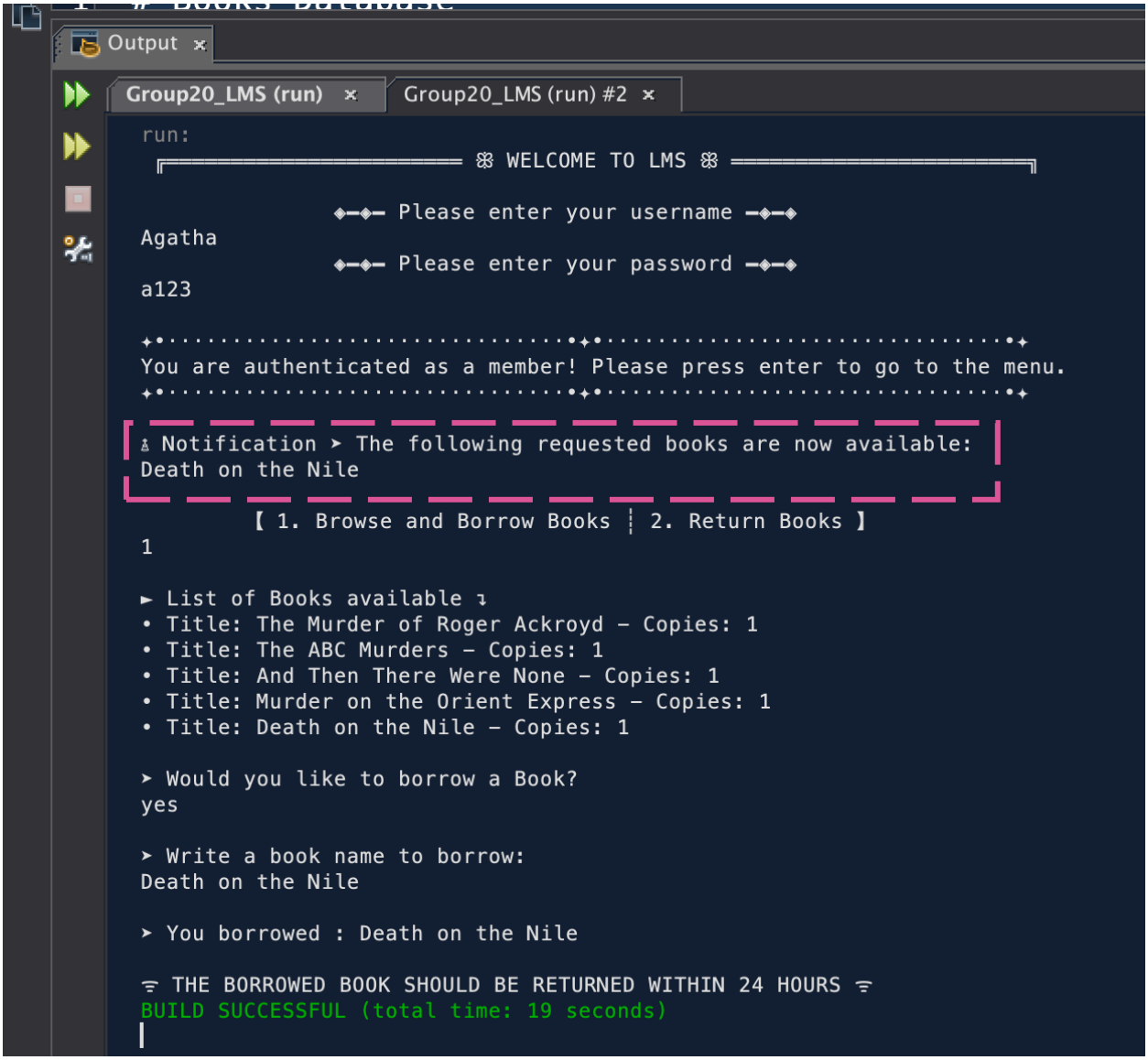


Figure 23: Book availability notification

After another member has returned a book requested by Agatha, Agatha logs into the Library Management System (LMS) and receives a notification. The notification informs Agatha that the previously requested book, "Death on the Nile," is now available. Agatha is then presented with a menu offering options to browse and borrow books, return books, or perform other library-related tasks. Agatha chooses to browse and potentially borrow books from the available selection. Upon browsing the books in the database, Agatha is presented with a list showing the available titles and their respective copies. The system then prompts Agatha to indicate if she would like to borrow a book.

This scenario demonstrates the LMS's capability to proactively notify users when requested items become available and provides a user-friendly interface for managing book transactions and interactions within the library system. Agatha can conveniently browse and borrow books based on her preferences and needs through the LMS interface.

5. Discussion

5.1 How the system meets the objectives set out in the introduction

1. Efficient Resource Management:

- **Client-Server Architecture:** The LMS employs a client-server model, with a central server managing book inventories and member transactions.
- **Database Management**: The server contains databases of registered users and available books, allowing libraries to manage inventory effectively by adding new books, deleting books, and tracking amounts.
- **Transaction Tracking:** The system keeps track of borrowing and return dates to ensure that book lending and availability are managed efficiently.

2. Secure Authentication and Access Control:

- **User Authentication:** Secure login approaches are used to authenticate users (librarians and members) before granting access to the LMS.
- **Role-Based Access Control:** Access control is based on user roles and permissions. Librarians have administrative authority to manage book inventory, whilst members have limited access to browse, borrow, and return books.

3. Real-time Information Updates:

- **Real-time Book Status:** The system gives real-time information on book availability and borrowing status. When members and librarians log in or engage with the system, they immediately obtain information regarding book availability and return dates.
- **Timely Notifications:** Members are notified about due dates and book availability, which encourages timely interactions with the LMS.

4. Reliable Communication Protocols:

- **TCP Protocol:** The usage of TCP guarantees that client apps and the central server communicate reliably and without errors.

5.2 Challenges faced and how they were addressed

- **Problem:**

The issue is that when a member borrows a book, other members may see unrelated books, leading to confusion and a less customized experience.

- **Solution:**

To address this issue, we will change the system to store separate lists for each member's borrowed or requested books. This guarantees that each member only sees books that they have borrowed or requested.

- **Implementation Steps:**

1. Create distinct lists for each user to store borrowed and requested books.
2. After borrowing a book, add it to the user's personal list of borrowed books. Similarly, when a user requests a book, add it in their list of wanted books.
3. Ensure that when a user checks in or views their borrowing and request history, only the books related with their account are displayed.
4. Use conditional checks in appropriate methods to ensure that processes, such as displaying borrowed books or processing borrowing requests, are customized to each user.

5.3 Potential improvements, additional features or future directions

1. Enhanced User Interface:

- **Modern User Interface (UI):** Create a more intuitive and visually appealing user interface for both the client application and web-based interfaces to help users (librarians and members) explore and engage with the system.
- **Responsive Design:** Make that the UI is responsive across all devices (desktop, tablet, and mobile) to support varying user preferences and usage circumstances.

2. Mobile App Support:

- **Mobile Applications:** Build specific mobile apps (iOS and Android) for the LMS, letting members browse, borrow, and return books using their smartphones or tablets.

3. Advanced Search and Recommendation System:

- **Advanced Search:** Use advanced search features such as keyword search, filters (by genre, author, publication year), and sorting choices to improve book discovery and browsing experiences.
- **Recommendation Engine:** Apply a recommendation engine based on user preferences and borrowing history to provide tailored book suggestions.

6. Conclusion

In conclusion, our Library Management System (LMS) project utilizes a client-server architecture to facilitate seamless interaction between users and the central server. Librarians wield administrative control over the book inventory, managing entries through tasks such as addition, removal, and modification. Members, conversely, have the ability to peruse available books, borrow them, and subsequently return them, with the system meticulously overseeing borrowing limits and tracking return dates.

To uphold secure access and preserve data integrity, robust authentication procedures are implemented, permitting access exclusively to authorized individuals based on their assigned roles. Real-time updates on book availability and borrowing statuses enhance the user experience by providing timely information to both librarians and members. These enhancements, coupled with reliable communication protocols like TCP, significantly enhance the overall efficiency and effectiveness of the LMS, optimizing library operations and fostering increased user engagement.

7. References

[1] TheKnowledgeAcademy, “What is client-server architecture? definition, types, & example,” What is Client-Server Architecture? Definition, Types, & Example, https://www.theknowledgeacademy.com/blog/client-server-architecture/.

[2] What is TCP/IP? | cloudflare, https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/ .

[3] “Draw.io - free flowchart maker and diagrams online,” Flowchart Maker & Online Diagram Software, https://app.diagrams.net/.

8. Appendixes

8.1 Source Code

Server Class:

```
package group20_lms;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

public class Server {

    // Users and Books data
    private static Map<String, String> users = new HashMap<>();
    private static Map<String, String> userRoles = new HashMap<>();
    private static Map<String, Integer> books = new HashMap<>();
    private static ArrayList<String> requestedBooks1 = new ArrayList<String>();
    private static ArrayList<String> requestedBooks2 = new ArrayList<String>();
    private static ArrayList<String> BooksBorrowed1 = new ArrayList<String>();
    private static ArrayList<String> BooksBorrowed2 = new ArrayList<String>();
    private static ArrayList<String> returnedBooks = new ArrayList<String>();

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1091);
        System.out.println("◉ ✓ Server on...\n");

        readUsers("users.txt");
        readBooks("books.txt");

        while (true) {
            System.out.println("◉ Server is waiting for a connection.");

            // accept incoming client connections
            Socket clientSocket = serverSocket.accept();

            // indicate successful client connection
            System.out.println("◉ ✓ Client Successfully Connected!");

            // handle client connection in a separate thread
            new ClientHandler(clientSocket).start();
        }
    }
```

```java
// Inner class to handle each client connection separately
private static class ClientHandler extends Thread {

    private Socket clientSocket;

    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }

    public void run() {

        try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
             PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

            // Print msg message
            out.println(" ╔═══════════════════ ❀ WELCOME TO LMS ❀ ═══════════════════╗");
            // Authenticate user
            String username = in.readLine();
            String password = in.readLine();

            // Check user info from database
            if (users.containsKey(username) && users.get(username).equals(password)) {
                String role = userRoles.get(username);
                out.println("You are authenticated as a " + role + "! Please press enter to go to the menu.");

                // If user is Librarian > display librarian menu
                if ("librarian".equalsIgnoreCase(role)) {
                    out.println(" 【 1. Add a new book ┆ 2. Update book information ┆ 3. Delete a book 】 ");
                    int option = Integer.parseInt(in.readLine());
                    switch (option) {
                        case 1:
                            out.println("➤ Enter book title:");
                            String title = in.readLine();
                            out.println("➤ Enter number of copies:");
                            int copies = Integer.parseInt(in.readLine());
                            AddBook(out, in, "books.txt", title, copies);
                            break;
                        case 2:
                            UpdateBook(out, in);
                            break;
                        case 3:
                            DeleteBook(out, in);
                            break;
                        default:
                            out.println("Invalid option.");
                            break;
                    }
```

```
                    // If user is Member > display member menu
                } else if ("member".equalsIgnoreCase(role)) {

                    // Send return date notification for user if he borrowed a book
                    sendNotifications(out,username);
                    // Send notification for availability of requested book if user chose to be notified
                    requestedBooks(out, in, returnedBooks, username);

                    out.println("\t【 1. Browse and Borrow Books ¦ 2. Return Books 】");
                    // Prompt Member to enter option
                    int option = Integer.parseInt(in.readLine());

                    // Cases of member
                    switch (option) {
                        // Browse & Borrow book
                        case 1:
                            out.println("➢ List of Books available ↴");
                            BrowseBook(out, in);
                            out.println("➢ Would you like to borrow a Book? ");
                            String optionBorrow = in.readLine();
                            if (optionBorrow.equals("yes")) {
                                out.println("➢ Write a book name to borrow: ");
                                String bookToBorrow = in.readLine();
                                BorrowBook(out, in, bookToBorrow, username);
                            } else {
                                out.println("Understood, Have a nice day! ");
                                break;

                            }

                        // Return book
                        case 2:
                            out.println("➢ Write a book name to return: ");
                            String bookToReturn = in.readLine();
                            ReturnBook(out, in, bookToReturn, username);
                            // Return method
                            break;
                        default:
                            out.println("Invalid option.");
                            break;
                    }
                }
            } // If user info is not on the database
            else {
                out.println("Authentication failed.");
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    // AddBook: a librarian method, to add new book to the database.
    private void AddBook(PrintWriter out, BufferedReader in, String filePath, String title, int copies) throws
IOException {
        BufferedWriter bw = new BufferedWriter(new FileWriter(filePath, true));
        if (!books.containsKey(title)) {
            bw.write(title + " " + copies);
            bw.newLine();
            bw.flush();
            books.put(title, copies);
            out.println("✔ Book added successfully ✔");
        } else {
            out.println("Book already exists.");
        }
    }
```

```java
// UpdateBook: a librarian method, to update book info > num of copies.
private void UpdateBook(PrintWriter out, BufferedReader in) throws IOException {
    out.println("➤ Enter book title to update:");
    String title = in.readLine();

    if (books.containsKey(title)) {
        out.println("➤ Enter new number of copies:");
        int newCopies = Integer.parseInt(in.readLine());

        books.put(title, newCopies);

        rewriteBooksToFile("books.txt");

        out.println("✔ Book updated successfully ✔");
    } else {
        out.println("Book not found.");
    }
}

// DeleteBook: a librarian method, to delete a specific book.
private void DeleteBook(PrintWriter out, BufferedReader in) throws IOException {
    out.println("➤ Enter book title to delete:");
    String title = in.readLine();

    if (books.containsKey(title)) {
        // Remove the book from the in-memory map
        books.remove(title);

        // Update the books.txt file to reflect the deletion
        rewriteBooksToFile("books.txt");

        out.println("✔ Book deleted successfully ✔");
    } else {
        out.println("Book not found.");
    }
}

// BrowseBook: a member method, to display a list of available books and browse them.
private void BrowseBook(PrintWriter out, BufferedReader in) throws IOException {
    int count = 0;
    for (Map.Entry<String, Integer> entry : books.entrySet()) {
        if (entry.getValue() != 0) {
            count++;
        }
    }

    out.println(count);  // Send the number of books to the client
    for (Map.Entry<String, Integer> entry : books.entrySet()) {
        if (entry.getValue() != 0) {
            out.println("• Title: " + entry.getKey() + " - Copies: " + entry.getValue());
        }
    }
}
```

```java
        // requestedBooks: a member method, to send notification for availability of requested books.
        private void requestedBooks(PrintWriter out, BufferedReader in, ArrayList<String> returnedBooks, String
username) throws IOException {
            ArrayList<String> availableBooks1 = new ArrayList<>();
            ArrayList<String> availableBooks2 = new ArrayList<>();
            int countAvailable1 = 0; // Correctly count the number of borrowed books
            int countAvailable2 = 0;

            if (username.equals("Sherlock")) {
                for (String book : returnedBooks) {
                    if (requestedBooks1.contains(book)) {
                        availableBooks1.add(book);
                        countAvailable1++;
                    }
                }
                // Send the count of available books to the client
                out.println(countAvailable1);

                // Notify the user about the availability of requested books
                if (countAvailable1 > 0) {
                    out.println("🔔 Notification ➤ The following requested books are now available:");
                    for (String availableBook : availableBooks1) {
                        out.println(availableBook);
                    }
                    // Clear the list of requested books since they have been notified
                    requestedBooks1.removeAll(availableBooks1);
                }
            } else {
                for (String book : returnedBooks) {
                    if (requestedBooks2.contains(book)) {
                        availableBooks2.add(book);
                        countAvailable2++;
                    }
                }
                // Send the count of available books to the client
                out.println(countAvailable2);

                // Notify the user about the availability of requested books
                if (countAvailable2 > 0) {
                    out.println("🔔 Notification ➤ The following requested books are now available:");
                    for (String availableBook : availableBooks2) {
                        out.println(availableBook);
                    }
                    // Clear the list of requested books since they have been notified
                    requestedBooks2.removeAll(availableBooks2);
                }

            }
        }
```

```
        // BorrowBook: a member method, to borrow a book and to be notified if the book is not instock.
        private void BorrowBook(PrintWriter out, BufferedReader in, String bookToBorrow, String username) throws
IOException {
            boolean bookFound = false;
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getKey().equalsIgnoreCase(bookToBorrow)) {
                    bookFound = true;
                    int copies = entry.getValue();
                    if (username.equals("Sherlock")) {
                        if (copies == 0) {
                            out.println("Book is not available ✘");
                            out.println("➤ Would you like to be notified when the book is in stock?");
                            String notifyUser = in.readLine();
                            if (notifyUser.equalsIgnoreCase("yes")) {
                                out.println("✔ You will be notified when the book is in stock ✔");
                                requestedBooks1.add(bookToBorrow);

                            }
                        } else {
                            books.put(entry.getKey(), copies - 1); // Update the number of copies
                            out.println("➤ You borrowed : " + entry.getKey());
                            out.println("᯽ THE BORROWED BOOK SHOULD BE RETURNED WITHIN 24 HOURS ᯽");
                            BooksBorrowed1.add(bookToBorrow);
                            break;
                        }
                    } else {
                        if (copies == 0) {
                            out.println("Book is not available ✘");
                            out.println("➤ Would you like to be notified when the book is in stock?");
                            String notifyUser = in.readLine();
                            if (notifyUser.equalsIgnoreCase("yes")) {
                                out.println("✔ You will be notified when the book is in stock ✔");
                                requestedBooks2.add(bookToBorrow);

                            }
                        } else {
                            books.put(entry.getKey(), copies - 1); // Update the number of copies
                            out.println("➤ You borrowed : " + entry.getKey());
                            out.println("᯽ THE BORROWED BOOK SHOULD BE RETURNED WITHIN 24 HOURS ᯽");
                            BooksBorrowed2.add(bookToBorrow);
                            break;
                        }

                    }
                    break; // Exit the loop once the book is found and processed
                }
            }
            rewriteBooksToFile("books.txt");
            if (!bookFound) {
                out.println("Book not found in the library ✘");
            }
        }

        // ReturnBook: a member method, to return books.
        private void ReturnBook(PrintWriter out, BufferedReader in, String bookToReturn, String username) throws
IOException {
            boolean bookFound = false;
            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                if (entry.getKey().equalsIgnoreCase(bookToReturn)) {
                    bookFound = true;
                    int copies = entry.getValue();
                    books.put(entry.getKey(), copies + 1); // Correctly update the number of copies
                    out.println("➤ You Returned : " + entry.getKey());
                    if (username.equals("Sherlock")) {
                        BooksBorrowed1.remove(bookToReturn);
                        returnedBooks.add(bookToReturn);
                    } else {
                        BooksBorrowed2.remove(bookToReturn);
                        returnedBooks.add(bookToReturn);
                    }
                    break;
                }
            }
            rewriteBooksToFile("books.txt"); // Move file writing outside the loop to avoid redundancy
            if (!bookFound) {
                out.println("Book not found in the library ✘");
            }
        }
```

```
    // sendNotifications: a member method, to send return date notification for user borrowed books.
    private void sendNotifications(PrintWriter out, String username) {

        if (username.equals("Sherlock")) {
            int counter1 = BooksBorrowed1.size(); // Count the number of borrowed books
            out.println(counter1); // Send the number of borrowed books to the client

            // Notification for borrowed books return dates
            if (!BooksBorrowed1.isEmpty()) {
                out.println("🔔 Notification ➤ Your borrowed books return dates:");
                for (String borrowedBook : BooksBorrowed1) {
                    String borrowedReturnDate = getReturnDateForBook(borrowedBook); // Use Date type
                    out.println(borrowedBook + " - Return Date: " + borrowedReturnDate);
                }
            }
        } else {
            int counter2 = BooksBorrowed2.size(); // Count the number of borrowed books
            out.println(counter2); // Send the number of borrowed books to the client
            if (!BooksBorrowed2.isEmpty()) {
                out.println("🔔 Notification ➤ Your borrowed books return dates:");
                for (String borrowedBook : BooksBorrowed2) {
                    String borrowedReturnDate = getReturnDateForBook(borrowedBook); // Use Date type
                    out.println(borrowedBook + " - Return Date: " + borrowedReturnDate);
                }
            }

        }

    }

    // Method to get return date for a borrowed book
    private String getReturnDateForBook(String bookName) {
        Calendar calendar = Calendar.getInstance();
        calendar.add(Calendar.HOUR_OF_DAY, 24); // 24 hours from now
        Date returnDate = calendar.getTime();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String returnDateString = dateFormat.format(returnDate);

        return returnDateString;
    }

}
```

```
    // Update database info
    private static void rewriteBooksToFile(String filePath) {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {
            writer.write("# Books Database");
            writer.newLine();
            writer.write("# Format: title,author,copies_available");
            writer.newLine();
            writer.newLine();

            for (Map.Entry<String, Integer> entry : books.entrySet()) {
                String title = entry.getKey();
                int copies = entry.getValue();
                writer.write(title + " " + copies);
                writer.newLine();
            }

            System.out.println("Books database updated successfully.");
        } catch (IOException e) {
            System.out.println("Error occurred while rewriting books database: " + e.getMessage());
        }
    }

    private static void readUsers(String filePath) throws FileNotFoundException, IOException {
        BufferedReader br = new BufferedReader(new FileReader(filePath));
        String line;
        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (!line.startsWith("#") && !line.isEmpty()) {
                String[] parts = line.split("\\s+");
                String username = parts[0];
                String password = parts[1];
                String role = parts[2];

                users.put(username, password);
                userRoles.put(username, role);
            }
        }
    }

    private static void readBooks(String filePath) throws FileNotFoundException, IOException {
        BufferedReader br = new BufferedReader(new FileReader(filePath));
        String line;
        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (!line.startsWith("#") && !line.isEmpty()) {
                String[] parts = line.split("\\s+");
                String title = String.join(" ", parts).replaceAll("\\s\\d+$", "");
                int copies = Integer.parseInt(parts[parts.length - 1]);
                books.put(title, copies);
            }
        }
    }

}
```

Figure 24: Server class source code

Client Class:

```java
package group20_lms;

import java.io.*;
import java.net.Socket;
import java.util.Scanner;

public class Client {

    public static void main(String[] args) throws IOException {

        // Create input and output streams
        BufferedReader bufferedReader;
        PrintWriter pw;

        // Create a socket connection to the server
        Socket serverSocket = new Socket("localhost", 1091);

        // Initialize input and output streams with socket streams
        bufferedReader = new BufferedReader(new InputStreamReader(serverSocket.getInputStream()));
        pw = new PrintWriter(new OutputStreamWriter(serverSocket.getOutputStream()), true);

        // Create a scanner to read user input
        Scanner user = new Scanner(System.in);

        // Display welcome message from the server
        System.out.println(bufferedReader.readLine());
        System.out.println("");
        // Prompt the user for username
        System.out.println("\t       Please enter your username ");
        String username = user.nextLine();
        pw.println(username);

        // Prompt the user for password
        System.out.println("\t       Please enter your password ");
        String password = user.nextLine();
        pw.println(password);

        System.out.println("\n");
        // Read the server response 'User role'
        String serverResponse = bufferedReader.readLine();
        System.out.println(serverResponse);
        System.out.println("");
        if (serverResponse.contains("member")) {

            user.nextLine();
            // Read the length string sent by the server
            String lengthString = bufferedReader.readLine();

            // Parse the length string into an integer
            int numOfBorrowed = Integer.parseInt(lengthString);

            // Notification for return date of client if he borrowed a book.
            if (numOfBorrowed >= 1) {
                // notifi
                serverResponse = bufferedReader.readLine();
                System.out.println(serverResponse);
                //return dates
                for (int i = 0; i < numOfBorrowed; i++) {
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                }
            }
```

```
// Read the length string sent by the server
String lengthStringAva = bufferedReader.readLine();

// Parse the length string into an integer
Integer numOfAva = Integer.parseInt(lengthStringAva);

// Notification for availability if client requested a book
if (numOfAva >= 1) { //2
    // Notification: The following requested books are now available:
    serverResponse = bufferedReader.readLine();
    System.out.println(serverResponse);
    // Available books list
    for (int i = 0; i < numOfAva; i++) {

        serverResponse = bufferedReader.readLine();
        System.out.println(serverResponse);

    }
    // Browse
    System.out.println("");
    serverResponse = bufferedReader.readLine();
    System.out.println(serverResponse);

} else {   // Browse
    System.out.println("");
    serverResponse = bufferedReader.readLine();
    System.out.println(serverResponse);

}

int selection = Integer.parseInt(user.nextLine());
pw.println(selection);
System.out.println("");

switch (selection) {
    case 1:

        serverResponse = bufferedReader.readLine();
        System.out.println(serverResponse);
        int numberOfBooks = Integer.parseInt(bufferedReader.readLine());

        for (int i = 0; i < numberOfBooks; i++) {
            serverResponse = bufferedReader.readLine();
            System.out.println(serverResponse);
        }

        System.out.println("");
        // Would you like to borrow a Book?
        serverResponse = bufferedReader.readLine();
        System.out.println(serverResponse);

        // yes
        String answer = user.nextLine();
        pw.println(answer);
```

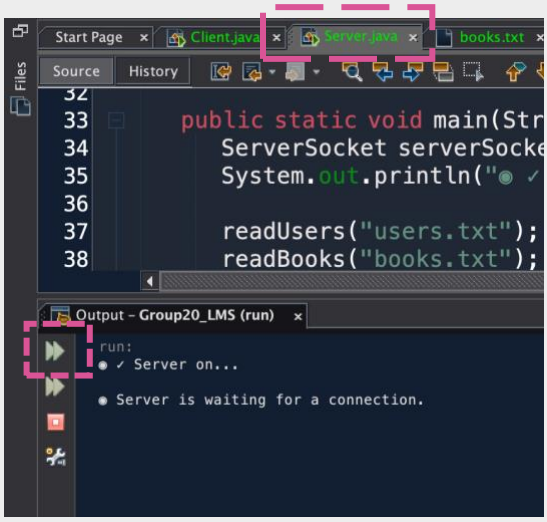
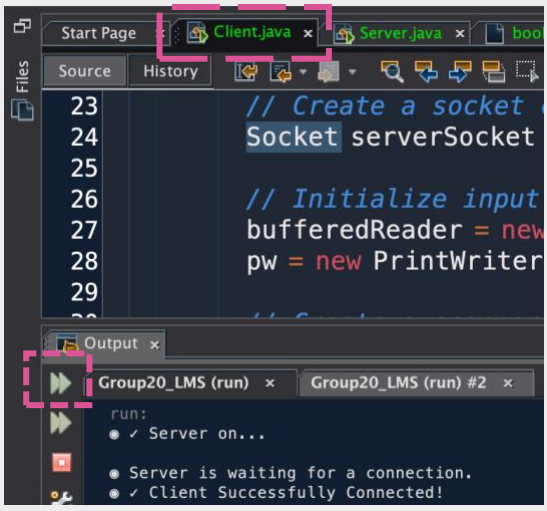
```
                String answer2 = "";
                if (answer.equalsIgnoreCase("Yes")) {

                    System.out.println("");

                    // Write a book name to borrow:
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);

                    // send the book name ex: "The ABC Murders"
                    String bn = user.nextLine();
                    pw.println(bn);

                    System.out.println("");
                    // You borrowed : The ABC Murders | Book is not available
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    System.out.println("");

                    // * THE BORROWED BOOK SHOULD BE RETURNED WITHIN 24 HOURS *
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);

                    if (serverResponse.equalsIgnoreCase("➤ Would you like to be notified when the book is in
                                                        + stock?")) {
                        answer2 = user.nextLine();
                        pw.println(answer2);
                        System.out.println("");
                    }
                    if (answer2.equalsIgnoreCase("Yes")) {
                        //System.out.println("🔔 User chose to be notified 🔔");
                        // You will be notified when the book is in stock
                        serverResponse = bufferedReader.readLine();
                        System.out.println(serverResponse);
                        break;
                    } else if (answer2.equals("no")) {
                        //System.out.println("🔕 User chose NOT to be notified 🔕");
                        serverResponse = bufferedReader.readLine();
                        System.out.println(serverResponse);
                        break;
                    }

                } else {
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    break;
                }
                break;

            case 2:
                // Write a book name to return
                serverResponse = bufferedReader.readLine();
                System.out.println(serverResponse);

                // send the book name
                String bn = user.nextLine();
                pw.println(bn);

                // You Returned :
                serverResponse = bufferedReader.readLine();
                System.out.println(serverResponse);

                break;

            default:
                System.out.println("Invalid Option ✘");
                break;
        }
```

```
        } else if (serverResponse.contains("librarian")) {
            System.out.println("");
            serverResponse = bufferedReader.readLine();
            System.out.println(serverResponse);

            int selection = Integer.parseInt(user.nextLine());
            pw.println(selection);
            System.out.println("");

            switch (selection) {
                case 1: {

                    // Enter book title msg
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    // Input of title
                    String newBookTitle = user.nextLine();
                    pw.println(newBookTitle);
                    System.out.println("");

                    // Enter no. of copies msg
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    // Input no. of copies
                    int newBookCopies = Integer.parseInt(user.nextLine());
                    pw.println(newBookCopies);
                    System.out.println("");
                    pw.flush();
                    break;
                }
                case 2: {

                    // Enter book title to update msg
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    // Input book title
                    String updateBookTitle = user.nextLine();
                    pw.println(updateBookTitle);
                    System.out.println("");

                    // Enter new no. of copies msg
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    // Input no. of copies
                    int updateBookCopies = Integer.parseInt(user.nextLine());
                    pw.println(updateBookCopies);
                    System.out.println("");
                    pw.flush();
                    break;
                }
                case 3: {
                    // Enter book title to delete msg
                    serverResponse = bufferedReader.readLine();
                    System.out.println(serverResponse);
                    // Input of book title
                    String deleteBookTitle = user.nextLine();
                    pw.println(deleteBookTitle);
                    System.out.println("");
                    pw.flush();
                    break;
                }
                default: {
                    System.out.println("Invalid Option ✘");
                    break;
                }
            }

            // Receive and print the response from the server
            serverResponse = bufferedReader.readLine();
            System.out.println(serverResponse);
        }
    }

}
```

Figure 25: Client class source code

8.2 Users' Guide

Member Guide:

Steps	
Step 1: Initiate Connection • Run Server class first. • Then client class.	 *Figure 27: server run*  *Figure 26: Client run*

Step 2: **If you are a member**

- Enter your username.
- Enter your password.

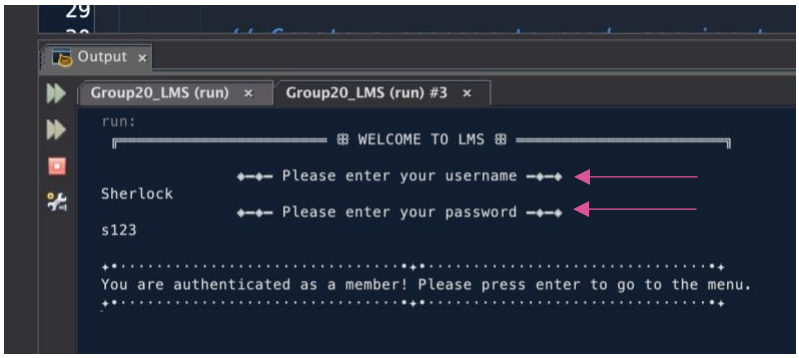


Figure 28: Member Login

Step 3: **Press enter to go to the menu.**

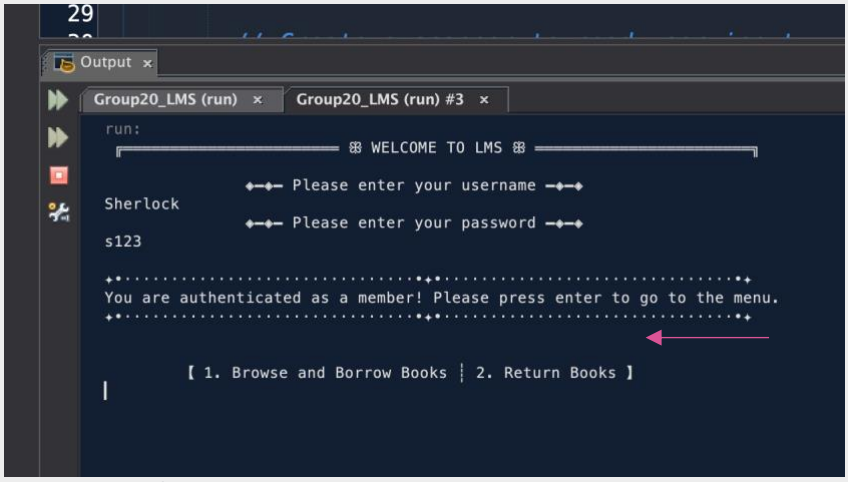


Figure 29: Member menu

Step 4: **Select your option**

- If you select 1, List of available books will be shown.

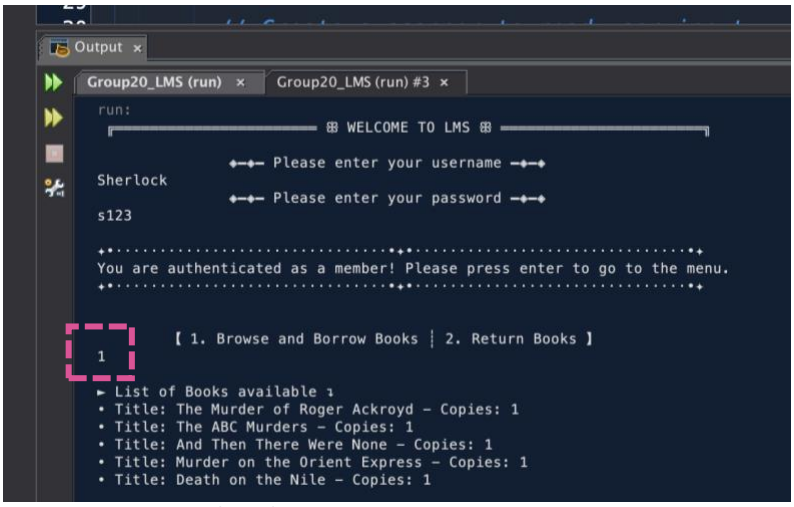


Figure 30: Option 1 selected

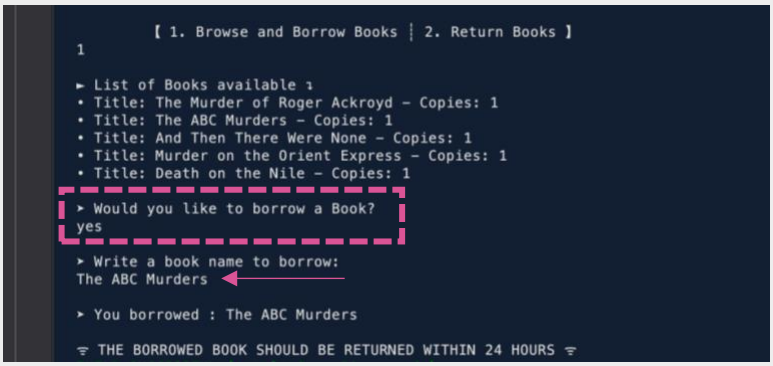
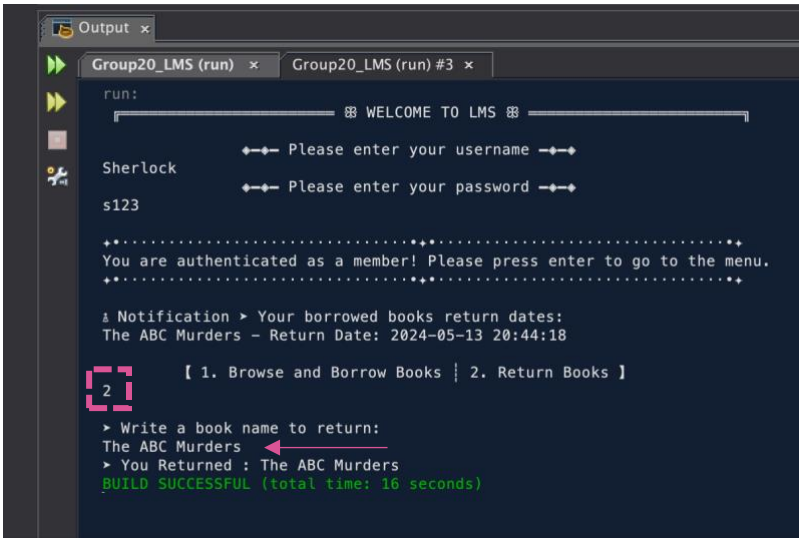
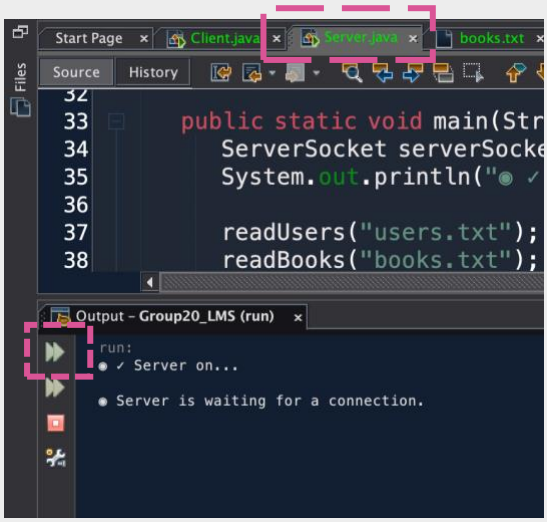
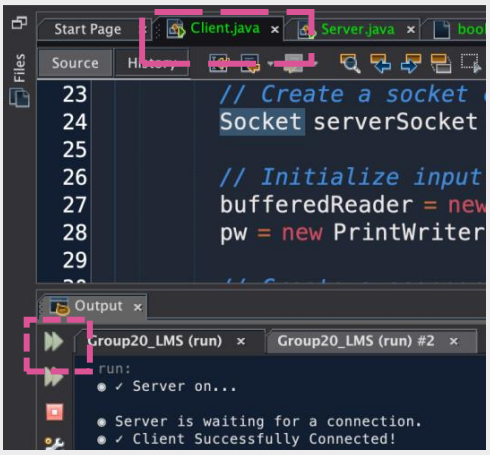
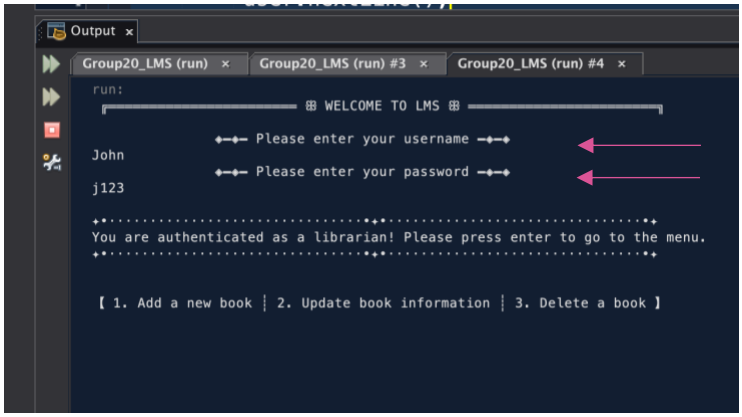
Step 5 (optional): Enter yes to Borrow a Book or no to exit • Enter the book title if you chose to borrow a book. Note *Borrowed books should be returned within 24 hours*	 *Figure 31: Borrow a book*
Step 6: Repeat all 4 steps to select another option • If you select 2, you will be able to return a book. • Write a book title to return. Note *When repeating, you don't have to run server class again just run client class*	 *Figure 32: Return a book*
End of Member Options	

Table 1: Member Guide

Librarian Guide:

Steps	
Step 1: Initiate Connection • Run Server class first. • Then client class.	 *Figure 33: Client run*  *Figure 34: Client run*
Step 2: If you are a Librarian • Enter your username. • Enter your password.	 *Figure 35: Librarian Login*

Step 3: Press enter to go to the menu.

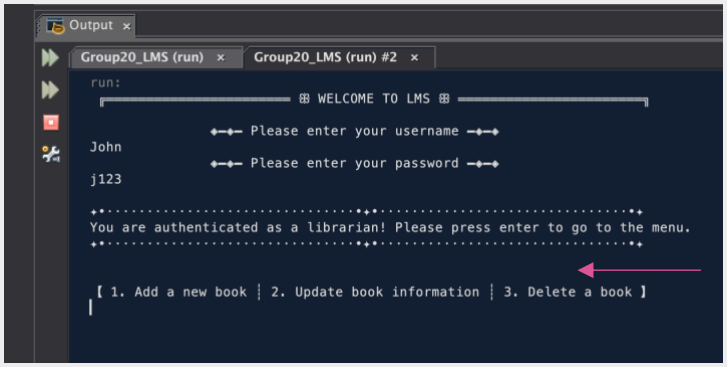


Figure 36: Librarian Menu

Step 4: Select your option

- If you select 1, you can add a new book to books collection.
- To add new book, enter book title.
- Then enter number of copies.

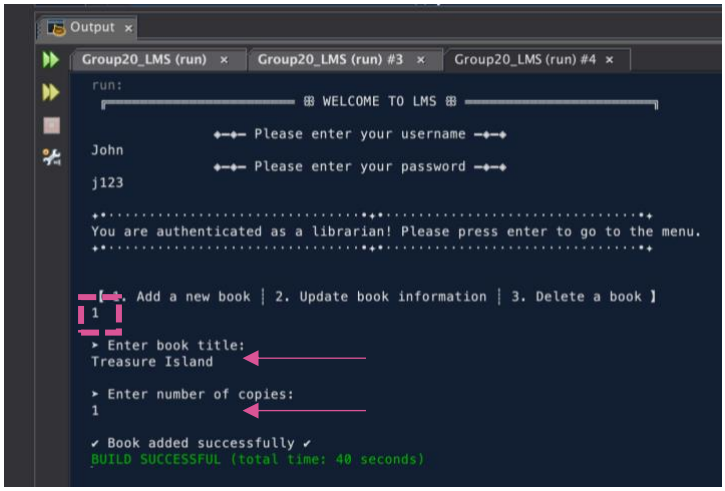


Figure 37: Option 1 selected, Add new book

Step 5: Repeat all 4 previous steps to select another option if needed.

- If you select 2, you'll be able to update book info (number of copies).
- To update info, enter book title first.
- Then the new number of copies.

Note *When repeating, you don't have to run server class again just run client class*

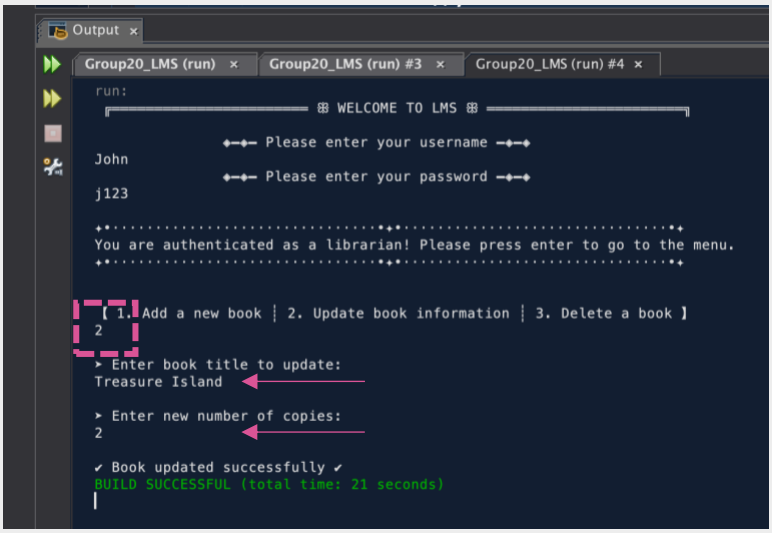


Figure 38: Option 2 selected, Update book info

Step 6: Repeat first 4 steps to select another option if needed.

- If you select 3, you will be able to delete a book from books collection.
- Write a book title to delete

Note *When repeating, you don't have to run server class again just run client class*

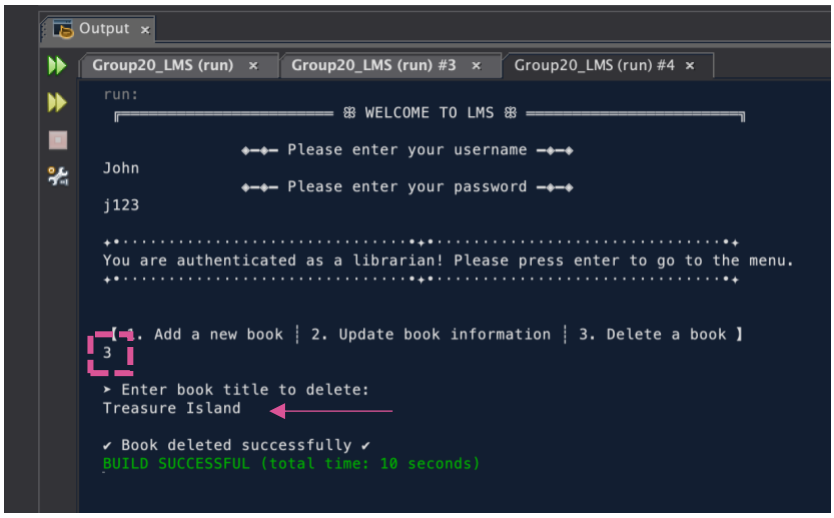


Figure 39: Option 3 selected, Delete a book

End of Librarian Options

Table 2: Librarian Guide