Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
"Library Management System"

[Code No.: COMP 202]

(For partial fulfillment of the requirements for internal assessment in Data Structure and Algorithm)

Submitted by
Rasik Poudel (037995-24)

Submitted to
Mr. Sagar Acharya
Department of Computer Science and Engineering

February 25, 2026

Acknowledgements

I would like to express my heartfelt gratitude to everyone who supported and contributed for the successful completion of my mini-project titled “Library Management System”.

I am deeply thankful to our lecturer, Mr. Sagar Acharya, for providing us with this golden opportunity through which I learned to apply what we learned in our Data Structure and Algorithm (DSA) lectures.

I would like to extend my thank to my classmates, friends, and families for their moral support and motivation during this mini-project.

Abstract

The Library Management System is a desktop application developed using C++ and the Qt framework to showcase the real-world application of basic data structures and Object Oriented Programming (OOP). The project combines basic structures such as Linked List, Hash Table, and Queue to create a fully functional graphical application for efficiently managing books.

A customized singly linked list is employed to dynamically store book information, with each book holding information such as name, author, ISBN, and issue status. A hash table with separate chaining is also developed to support efficient book searches using ISBN as the key. Finally, a queue data structure with the First In , First Out (FIFO) rule is employed to support the waiting list of issued books.

The system supports operations such as adding books, issuing books, returning books, and searching books by International Standard Book Number (ISBN) or name using a fully functional graphical user interface developed using the Qt framework. The LibraryManager controller is responsible for ensuring proper interaction between the Graphical User Interface (GUI) and the data structure component of the system.

This project is a learning aid that connects theoretical knowledge of data structures to real-world application development, improving comprehension of dynamic memory allocation, algorithm development, and software design.students seeking to understand recursion, stack operations, and algorithmic problem solving through visualization.

Contents

List of Figures

Abbreviations

CPU Central Processing Unit.

DSA Data Structure and Algorithm.

FIFO First In , First Out.

GUI Graphical User Interface.

IDE Integrated Development Environment.

ISBN International Standard Book Number.

OOP Object Oriented Programming.

OS Operating System.

STL Standard Template Library.

Chapter 1

Introduction

1.1 Background

Library management plays an important role in organizing and maintaining book records efficiently. Traditional manual systems are time-consuming and prone to errors, especially when managing large collections of books. A digital Library Management System improves accuracy, efficiency, and accessibility of records.

The Library Management System developed in this project is a C++ based desktop application built using the Qt framework. It integrates fundamental data structures such as Linked List, Hash Table, and Queue to manage books and waiting lists effectively. The system allows users to add books, issue books, return books, and search for books through an interactive graphical user interface.

By combining data structures with GUI development, the project demonstrates how theoretical concepts can be applied in real-world software applications.

1.2 Objectives

- To develop a GUI-based Library Management System using C++ and Qt.
- To implement core data structures such as Linked List, Hash Table, and Queue manually.
- To enable efficient book management operations including add, issue, return, and search.
- To demonstrate the practical application of OOP.
- To strengthen understanding of DSA through real-world implementation.

1.3 Motivation and Significance

Many students learn data structures theoretically but struggle to understand how they are applied in practical real world systems. This project is motivated by the idea of bridging that gap by building an interactive application using fundamental data structures.

Through this system:

- Students can observe how Linked Lists store dynamic records.
- Hash Tables improve searching efficiency.
- Queues manage waiting lists using the FIFO principle.

Educational Value: Provides hands-on experience in implementing and integrating data structures with /glsGUI applications.

Conceptual Clarity: Demonstrates how different data structures work together within a single system.

Practical Skill Development: Enhances skills in C++ programming, dynamic memory management, and Qt-based GUI development.

This project transforms theoretical data structure concepts into a structured and interactive software solution suitable for academic learning.

Chapter 2

Related Works

Library Management Systems have been widely developed and studied in academic and practical environments as a common application for demonstrating OOP and DSA concepts. Many basic implementations are available in textbooks, online tutorials, and programming learning platforms where the system is often built using arrays, files, or database-driven approaches.

Several web-based and desktop applications provide digital library solutions using technologies such as Java, Python, and C++. Many of these systems depend heavily on database management systems (DBMS) for storing book records and user information. While such implementations are effective for real-world deployment, they often abstract away the internal working of fundamental data structures.

In educational platforms such as GeeksforGeeks and other programming tutorial websites, simplified console-based Library Management Systems are presented to illustrate operations like adding, deleting, issuing, and searching books. However, these implementations frequently utilize built-in data structures or standard template libraries, which limit deeper understanding of manual data structure implementation.

In comparison, the proposed Library Management System is implemented using C++ and the Qt framework, with a strong emphasis on custom-built data structures. A singly linked list is used for dynamic storage of book records, a hash table with separate chaining is implemented for efficient searching using ISBN as the key, and a queue structure manages waiting lists following the FIFO principle.

Unlike database-oriented solutions, this project focuses on demonstrating how multiple data structures can work together within a single application. By integrating GUI development with manual implementation of data structures, the system provides a practical learning model for students studying DSA and OOP. This makes the project particularly suitable as an academic mini project that emphasizes conceptual clarity and implementation skills rather than relying solely on external libraries or database systems.

Chapter 3

Literature Review

Library Management Systems are widely used in academic projects to demonstrate principles of OOP and DSA. Traditional implementations described in textbooks focus on managing book records using structured programming or database-driven approaches. Many introductory examples present console-based systems to teach file handling and basic record management.

Standard data structure literature highlights the importance of Linked Lists for dynamic storage, Hash Tables for efficient searching, and Queues for managing sequential access. Hashing techniques are commonly discussed for fast retrieval using unique identifiers such as ISBN, due to their average constant-time search performance. However, most academic examples explain these structures separately rather than integrating them into a unified application.

Online programming platforms such as GeeksforGeeks provide tutorials for building Library Management Systems in languages like C++, Java, and Python. While these resources effectively explain operations such as adding, deleting, issuing, and searching books, many rely on built-in libraries or database systems. This often abstracts away internal data structure implementation and memory management details.

Educational research suggests that implementing data structures manually improves conceptual clarity, particularly in understanding pointer manipulation, dynamic memory allocation, and collision handling in hash tables. Integrating these structures within a GUI framework such as Qt further enhances learning by combining backend logic with user interaction.

Overall, existing resources support the educational value of application-based learning, but many beginner-level systems depend heavily on external abstractions. This project addresses that gap by developing a C++ and Qt-based Library Management System that emphasizes manual implementation of Linked Lists, Hash Tables, and Queues, reinforcing fundamental DSA and OOP concepts within a single integrated application.

Chapter 4

Design and Implementation

4.1 System Architecture

The Library Management System is designed as a modular C++ application using GUI for the graphical user interface and custom implementations of core DSA. The system follows a layered architecture separating user interface, business logic, and data structure modules. The application runs on a standard Operating System (OS) and requires basic hardware resources such as Central Processing Unit (CPU) and memory for execution. Development was performed using an Integrated Development Environment (IDE) with CMake build configuration.

The architecture consists of the following major components:

4.1.1 User Interface Module (Qt GUI)

- Provides interactive windows for user operations.
- Allows adding, deleting, searching, issuing, and returning books.
- Displays system messages and validation warnings.
- Handles user input and forwards requests to the controller.

4.1.2 Controller Module

- Acts as a bridge between GUI and data structure modules.
- Implements core library operations.
- Ensures business rules such as preventing duplicate ISBN.

4.1.3 Data Structure Module

- **Linked List:** Stores book records dynamically.
- **Hash Table:** Enables fast searching using ISBN.
- **Queue:** Manages book issuing requests based on the FIFO principle.

4.2 System Design Overview

The system follows an object-oriented design approach using OOP principles:

- **Encapsulation:** Book attributes and operations are defined inside the Book class.
- **Modularity:** Separate classes for LinkedList, HashTable, Queue, and LibraryManager.
- **Abstraction:** The GUI interacts with high-level controller functions rather than directly manipulating data structures.

4.3 Data Structure Design

4.3.1 Book Class

Each book record contains:

- Title
- Author
- ISBN
- Availability status

4.3.2 Linked List

- Used for dynamic storage of book records.
- Supports insertion and deletion operations.
- Efficient for sequential traversal.

4.3.3 Hash Table

- Uses a hashing function based on ISBN.
- Provides fast searching capability.
- Handles collisions using chaining with a Linked List.

4.3.4 Queue

- Implements the FIFO principle.
- Used for managing issuing or waiting requests.

4.4 Implementation Details

The system is implemented in C++ using the Qt framework.

- The GUI is created using Qt Designer (.ui files).
- Signals and slots mechanism connects user actions to backend logic.
- Core data structures are manually implemented instead of using Standard Template Library (STL) containers.
- CMake is used for project configuration and compilation.
- The system operates entirely at runtime without database or file storage.

4.5 Workflow Algorithm

1. Start application.
2. Display proceed window.
3. Transfer control to the main window.
4. Initialize data structures (Linked List, Hash Table, Queue).
5. Display main window.
6. User selects an operation (Add/Search/Delete/Issue/Return).
7. Controller processes the request.
8. Data structure performs the required operation.
9. GUI updates the display accordingly.
10. End.

4.6 Error Handling and Validation

- Prevents duplicate ISBN entries.
- Displays warning messages for invalid inputs.
- Ensures book availability before issuing.
- Handles empty list conditions gracefully.

Chapter 5

System Overview

5.1 System Requirement Specifications

5.1.1 Software Requirements

This section describes the software components required to develop and run the Library Management System.

Software Requirements

- **Operating System:** Windows 10 / Linux OS
- **Programming Language:** C++
- **GUI Framework:** Qt (Qt Widgets, Qt Designer) GUI
- **Build System:** CMake
- **IDE / Code Editor:** Qt Creator / VS Code IDE

5.1.2 Hardware Specifications

This section lists the minimum hardware requirements needed to run the system.

Hardware Requirements

- **Processor:** Dual-core CPU
- **RAM:** 2 GB or higher
- **Storage:** 200 MB free space
- **Graphics Card:** Not required

5.2 Functional Requirements

The functional requirements define the operations supported by the Library Management System.

1. **Add Book:** The system shall allow the user to add new book records with title, author, ISBN, and availability status.
2. **Delete Book:** The system shall remove a book record using ISBN as the key.
3. **Search Book:** The system shall search for a book efficiently using a hash table based on ISBN.
4. **Issue Book:** The system shall issue books following the FIFO principle using a queue data structure.
5. **Return Book:** The system shall allow issued books to be returned and update availability status.
6. **Display Books:** The system shall display all stored book records using linked list traversal.
7. **Runtime Data Handling:** The system shall manage all data during program execution without permanent storage.

Chapter 6

Results and Discussion

6.1 Results

The Library Management System was successfully implemented using C++ and the Qt framework. The system performed all intended operations correctly during runtime using manually implemented data structures, without relying on STL containers, file storage, or databases.

The linked list dynamically stored all book records and supported efficient traversal and deletion. The hash table enabled fast book searching using ISBN as the key, with collision handling implemented through chaining. The queue correctly followed the FIFO principle for managing book issuing operations.

All core functionalities—adding books, deleting books, searching by ISBN and title, issuing books, returning books, and displaying book records—worked as expected through the GUI. User interactions were handled using Qt's signal and slot mechanism, and system feedback was provided using message dialogs.

The system was tested with multiple book entries and repeated operations. In all test cases, the custom data structures behaved correctly during runtime. However, all data was lost after application termination, as no persistent storage mechanism was implemented.

6.2 Discussion

The results demonstrate that fundamental DSA concepts such as linked lists, hash tables, and queues can be effectively integrated into a real-world application using C++ and Qt. The separation of GUI, controller, and data structure layers improved code clarity and maintainability.

The hash table significantly improved search efficiency compared to linear traversal, while the linked list provided flexibility for dynamic memory management. The queue reinforced understanding of FIFO behavior through practical book issuing operations.

The absence of file storage and database support was a deliberate design decision to maintain focus on core DSA concepts rather than system persistence. This aligns with the academic objectives of the mini project.

Overall, the project successfully met its goals by demonstrating the practical application of data structures and object-oriented programming concepts within a GUI-based system.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

This mini-project successfully implemented a Library Management System using C++ and the Qt framework with a strong focus on DSA concepts. The system demonstrated how fundamental data structures can be applied in a real-world scenario through a GUI-based application.

Custom implementations of linked list, hash table, and queue were used to manage book records dynamically during runtime. The linked list handled storage and traversal, the hash table enabled efficient searching using ISBN, and the queue followed the FIFO principle for book issuing operations. This design helped strengthen practical understanding of core data structures without relying on STL containers.

The separation of the GUI, controller, and data structure layers improved code clarity and modularity. Although the system does not support permanent data storage, it successfully met the academic objectives of the mini project by demonstrating object-oriented programming and data structure integration within a functional application.

7.2 Future Works

The current Library Management System serves as a foundational implementation. Several enhancements can be considered in future versions to extend its functionality:

- **File Persistence:** Add file handling to store and reload book data between executions.
- **User Roles:** Introduce admin and user roles with different access permissions.
- **Advanced Search:** Extend searching to include author and partial title matching.
- **UI Improvements:** Enhance the GUI with improved layouts, icons, and table-based views.

- **Data Validation:** Add stronger input validation to prevent duplicate or invalid ISBN entries.

References

cppreference.com (2024). C++ reference documentation. https://en.cppreference.com/. Accessed: Feb 23, 2026.

GeeksforGeeks (2024a). Hashing data structure. https://www.geeksforgeeks.org/hashing-data-structure/. Accessed: Feb 24, 2026.

GeeksforGeeks (2024b). Linked list data structure. https://www.geeksforgeeks.org/data-structures/linked-list/. Accessed: Feb 24, 2026.

GeeksforGeeks (2024c). Queue data structure. https://www.geeksforgeeks.org/queue-data-structure/. Accessed: Feb 24, 2026.

Qt Documentation (2024). Qt documentation: Widgets and signals & slots. https://doc.qt.io/. Accessed: Feb 23, 2026.

Appendix A

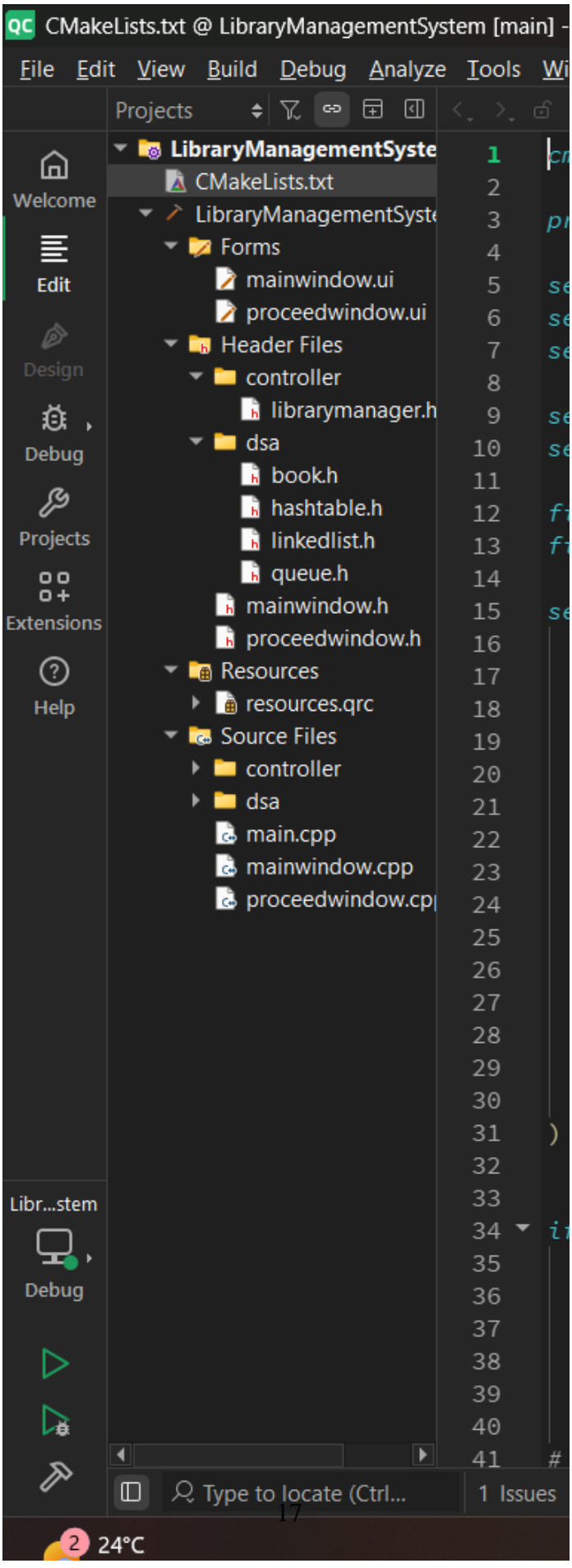

Figure 7.1: Structure of the program

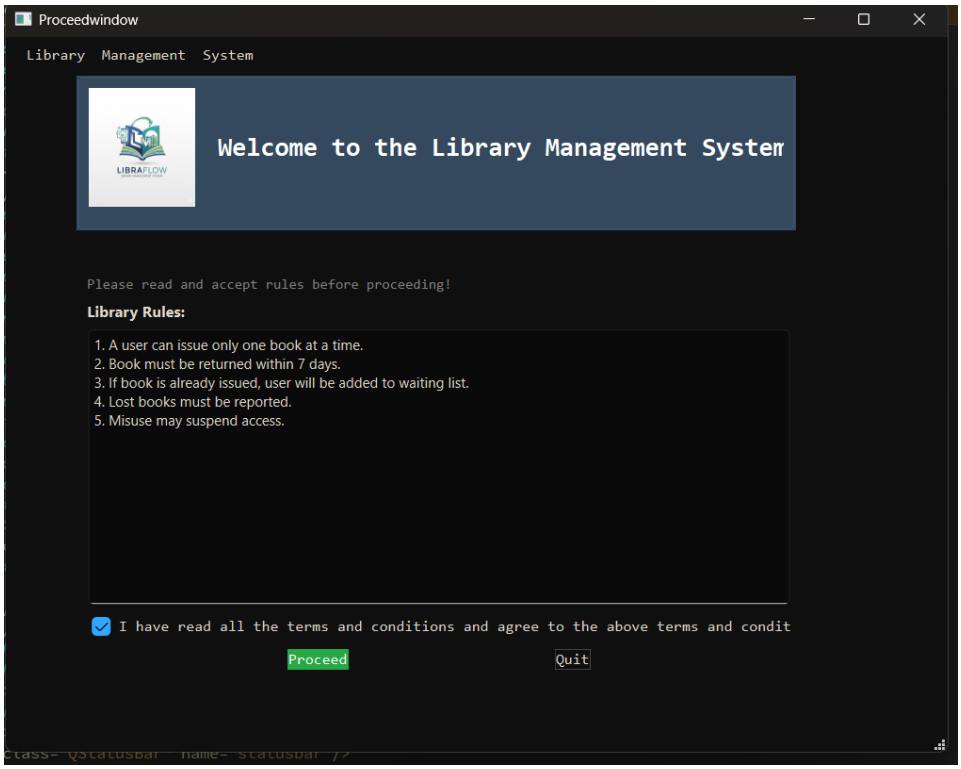

Figure 7.2: Proceed window

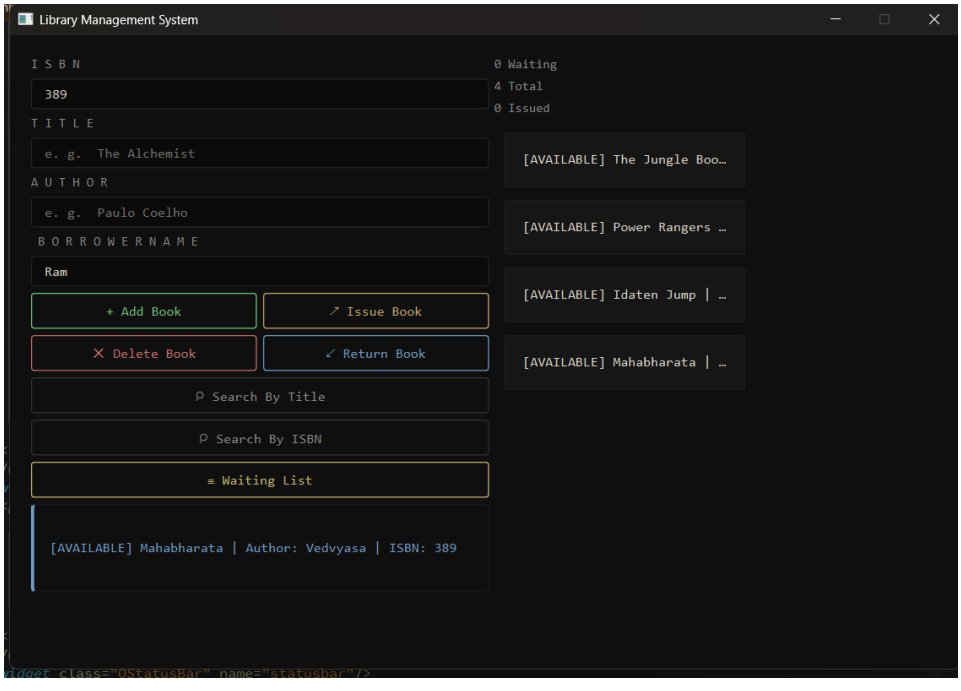

Figure 7.3: searching a book

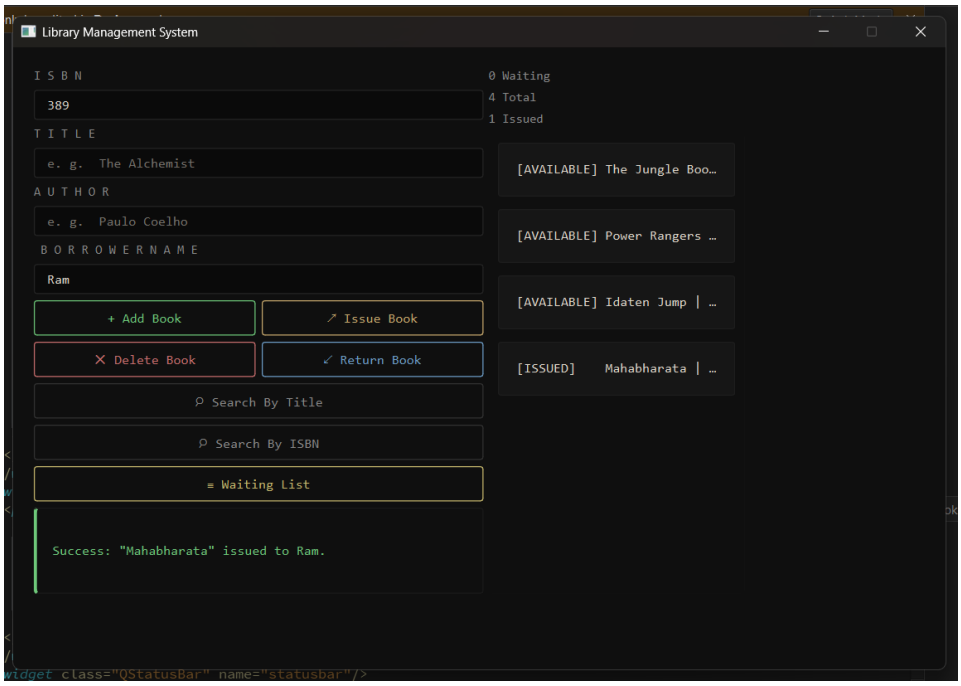

Figure 7.4: issuing a book

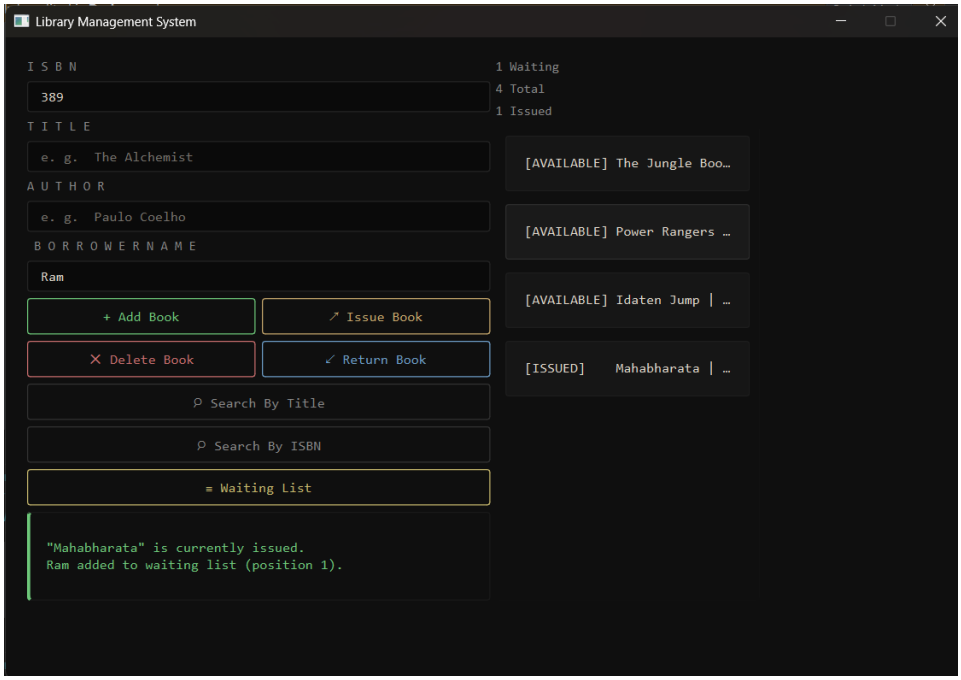

Figure 7.5: queue while reissuing a book