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 Year II / Semester I in Computer Engineering)

Submitted by
Bandana Gyawali (Roll No. 19)

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

February 24, 2026

Acknowledgements

I extend my sincere gratitude to Mr. Sagar Aacharya for his guidance and support during the development of this mini project. The concepts I learned in my Data Structures and Algorithms classes served as the essential foundation for developing this project, and the constructive feedback I received helped me refine my practical implementation. I also want to thank the teaching staff of the Department of Computer Science and Engineering and my institution for providing the resources and environment needed for this project. Finally, I appreciate my friends for their helpful discussions and my family for their encouragement, which helped me stay focused until the project was finished.

Abstract

This project presents a Library Management System which uses core concepts of Data Structures and Algorithms to improve how libraries manage their daily operations in a modern digital environment. The application employs a doubly linked list as its central data structure to store and manage book records, ensuring that adding, deleting, and searching for books is both fast and organized. The system includes several practical features, such as automatic location code generation, a live search tool, and a real-time fine calculation that track late returns. It also provides a summary dashboard to show the total number of books and their current status . Developed using C++ and the Qt framework, the project offers a clean and user-friendly interface for librarians. By combining fundamental data structures with modern software tools, the project demonstrates how theoretical DSA principles can be applied to solve real world management challenges while improving the overall efficiency of library services.

Keywords: *Data Structures and Algorithms, Doubly Linked List, Library Management System, Qt Framework, C++, Fine Calculation.*

Contents

List of Figures

Abbreviations

DLL	Doubly Linked List.
DSA	Data Structures and Algorithms.
GUI	Graphical User Interface.

Chapter 1

Introduction

This “Library Management System” is a simple desktop based application that works to help librarians manage book records and track book assignments to students smoothly. It allows users to add new books, assign them to students, process returns, and calculate late fees automatically. Automating library tasks plays a vital role in saving time and keeping data organized. This mini project, titled Library Management System, applies basic concepts of Data Structures and Algorithms (DSA) to design a system that stores book data and makes management easier. Using a Doubly Linked List (DLL) and C++, the project turns complex data ideas into a helpful application that simplifies the daily workload of a librarian.

1.1 Background

In the past, libraries had to track everything by hand using paper registers . These manual methods were very slow and it was easy to lose a record or make a mistake when calculating late fees. Even when early digital methods like basic spreadsheets came along, they still had many problems. These older digital systems were hard to search through and if you wanted to add or delete a book, the computer had to move every other piece of data . This project uses a Doubly Linked List to solve the problems found in those older systems. Unlike basic lists, a Doubly Linked List allows the software to add or remove book records instantly without needing to reorganize the whole memory. This modern approach, built with C++ and the Qt framework, uses DSA concepts for creating a fast and reliable system that makes the daily work of a librarian much easier.

1.2 Objectives

- To demonstrate the practical application of DSA concepts in solving real world library management challenges.
- To design and implement a book record model using a Doubly Linked List for dynamic storage and management of library data.
- To apply functions like traversal, addition, and deletion for managing the collection of books .
- To use unique book IDs for finding their location, managing the assignment and return of books to students with automatic fine calculation.

1.3 Motivation and Significance

The motivation for choosing this project was to use the linked list concepts I learned and apply them to solve a real life problem. After seeing how difficult it is to manage a library using past paper based and digital methods, I wanted to use to make the whole process faster and more organized. The significance of this work lies in demonstrating that even a relatively simple data structure, such as a linked list, is enough to build a useful and complete software system. It shows how creative thinking and simple data structures can significantly improve how a library functions.

Chapter 2

Related Works

2.1 Koha Open Source Library System

Koha is a professional software used by big libraries around the world to track books and members.

- Description: It is a web-based system that helps libraries keep a digital record of every book and student.
- Technologies: It uses a large SQL database to store thousands of records permanently and safely.
- Strengths: It is very powerful and can handle everything from tracking due dates to managing complex library inventory.
- Limitations: Because it is made for huge libraries, it is very complicated to set up and requires expert knowledge to run.
- Comparison: Koha is built for massive public libraries, whereas my project is a simple solution for smaller school libraries. While Koha uses heavy databases, my system uses a Doubly Linked List to keep the software fast and easy to use on a normal computer.

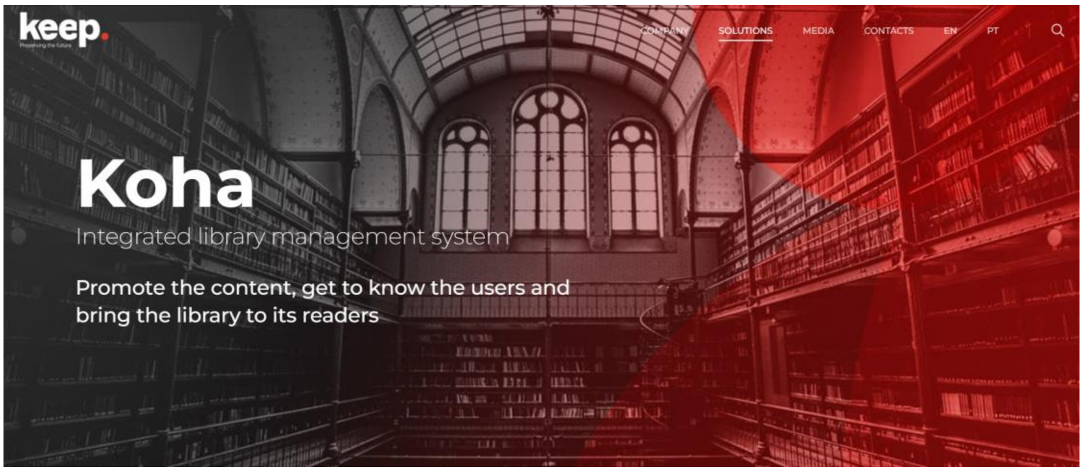

keep.
COMPANY
SOLUTIONS
MEDIA
CONTACTS
EN
PT
Koha
Integrated library management system
Promote the content, get to know the users and bring the library to its readers

Figure 2.1: Koha Open Source Library System

2.2 Evergreen Integrated Library System (ILS)

Evergreen is another highly advanced system designed to help large groups of libraries share their resources together.

- Description: It is a massive software used to manage catalogs and checkouts across many different library branches at once.
- Technologies: Built using a mix of Perl, C, and PostgreSQL to handle millions of tasks without slowing down.
- Strengths: It is excellent at resource sharing, allowing a student in one city to see if a book is available in another city library.
- Limitations: It is far too heavy for a single small library and needs a team of IT professionals to keep it running.
- Comparison: Evergreen focuses on connecting many libraries through a complex network. My project focuses on the efficiency of a single library by using a Doubly Linked List to make book management and fine calculation instant for a single user.

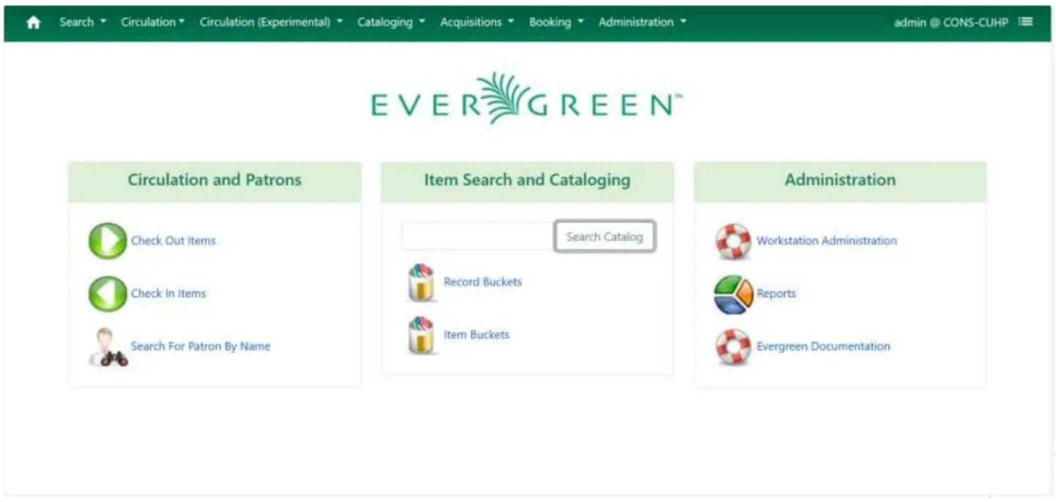


Figure 2.2: Evergreen Integrated Library System

Chapter 3

Design and Implementation

The design and implementation of this project was planned meticulously and was initiated so that every step was precise and meaningful to the proper development of the system.

3.1 System Overview

The library system is a desktop application designed to replace manual book tracking with a digital process. It uses smart data structures to organize library tasks into three main parts:

- Administrative Module: Handles the overall management of the library, including adding new books and removing old ones.
- Search and Location Feature: Uses unique book IDs to find exactly where a book is kept on the library shelves.
- Student and Fine Feature: Manages the process of giving books to students and calculating late fees automatically when books are returned.

3.2 System Requirement Specifications

3.2.1 Functional Requirements

These are the specific tasks the system must perform:

- Book Entry: The system must allow the librarian to enter book details like title, author, and should generate a unique location ID for each book.
- Data Linking: Every book must be stored in a doubly linked list to allow the user to move forward and backward through the records.
- Searching: The system must use the book ID to instantly find a specific book's location.
- Fine Calculation: The system must track how many days a book was kept and calculate a fine if it is returned late.

3.2.2 Non-Functional Requirements

These describe the quality of the system:

- Speed: Because it uses linked lists, the system should add and delete book records very quickly.
- Accuracy: The fine calculation must be exact to ensure fair treatment for all students.
- Easy to Use: The interface should be simple enough for a librarian to use without special technical training.
- Reliability: The system must keep book records safe in the computer's memory while the program is running.

3.2.3 Software Requirements

The tools used to build this library system include:

- Programming Language: C++ for the core logic and data management.
- Data Structure: A custom built Doubly Linked List used for dynamic storage and management of book records.
- Framework: Qt Widgets for creating the windows, buttons, and text boxes.
- Development Environment: Qt Creator was used as the main workspace for coding and design.

3.2.4 Hardware Requirements

- Processor: A minimum dual-core processor (e.g., Intel Core i3 or equivalent).
- Memory (RAM): A minimum of 4 GB RAM.
- Storage: At least 200 MB of available disk space.
- Display: A monitor with a minimum resolution of 1280×720 pixels.
- Input Devices: A standard keyboard and mouse.

3.3 System Design

3.3.1 Functional Workflow

The operational logic of the Library Management System is designed to mirror the daily activities of a physical library while automating complex tasks. Figure 3.1 illustrates the process flow from the initial dashboard to specific core functions.

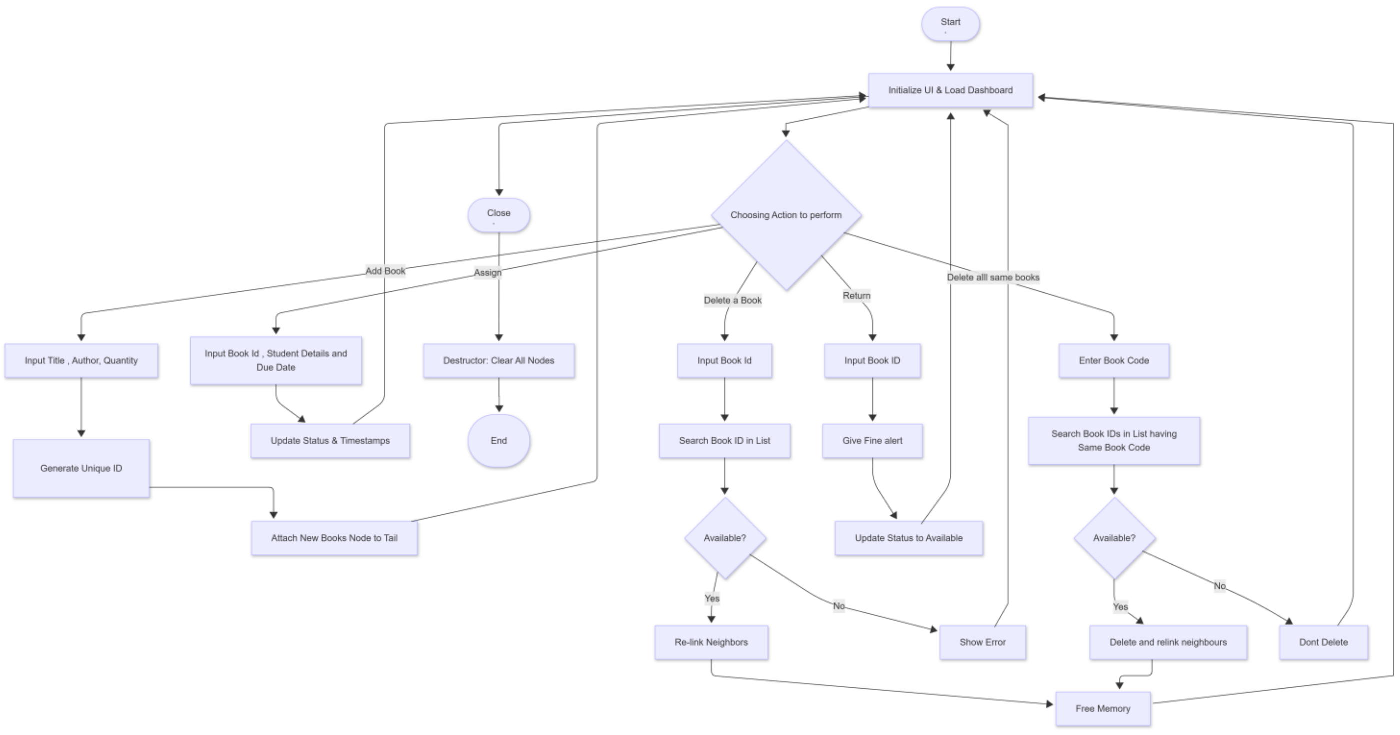

Start
Initialize UI & Load Dashboard
Choosing Action to perform
Close
Add Book
Assign
Delete a Book
Return
Delete alll same books
Input Title , Author, Quantity
Input Book Id , Student Details and Due Date
Destructor: Clear All Nodes
Input Book Id
Input Book ID
Enter Book Code
Generate Unique ID
Update Status & Timestamps
End
Search Book ID in List
Give Fine alert
Search Book IDs in List having Same Book Code
Attach New Books Node to Tail
Available?
Update Status to Available
Available?
Yes
No
Yes
No
Re-link Neighbors
Show Error
Delete and relink neighbours
Dont Delete
Free Memory

Figure 3.1: System WorkFlow

3.3.2 Data Structure Design

The backbone of this system is a custom-implemented Doubly Linked List. Each "Node" in the list represents a book, containing both the book's data and two pointers: *next* and *prev*. This design was chosen to allow for dynamic memory allocation so that the library can grow without a predefined limit. The use of both pointers allows the system to traverse the collection in both directions and enables efficient $O(1)$ deletion once a book is located, as the system can relink the surrounding nodes without rescanning the entire list.

3.4 Implementation Details

The implementation follows a modular C++ architecture where the Graphical User Interface (GUI) communicates with the backend logic through the Qt Signal and Slot mechanism. This ensures that whenever a change is made to the linked list (such as adding a new book), the user interface updates instantly to reflect the new state of the collection. A key implementation detail is the Smart location ID Generation logic. To prevent duplicate IDs after a book is deleted, the system does not simply count the nodes. Instead, it performs a search to find the highest existing serial number and increments it by one. Furthermore, memory safety is prioritized through a manual destructor that traverses the list and deletes every node when the application is closed, preventing memory leaks in the RAM.

3.5 Summary

By utilizing DSA , specifically a Doubly Linked List, this system provides a reliable way that help to manage a library. The design focuses on keeping data safe and making the library management easier by automating difficult tasks like fine calculation and record organization. Overall, the project turns difficult manual work into fast, digital process that any librarian can use with ease.

Chapter 4

Results and Discussion

The Library Management System was developed to improve the efficiency of managing book collections. While the doubly linked list model made data management dynamic and flexible, scaling to very large inventories can be further optimized in future versions. This project demonstrates how digital automation can replace manual record-keeping, reducing errors in tracking and fine collection. The system achieved its core objectives by delivering fast search results, a professional user interface, and accurate automated calculations.

4.1 System Features

- **Efficient Data Management and Insertion:** The system uses a doubly linked list to store book records, which provides a dynamic size compared to fixed arrays. Every book is a node with pointers to both its successor and predecessor. The use of a tail pointer ensures that adding a new book at the end of the collection is an efficient $O(1)$ operation.

- **Unique ID Generation and Location Tracking:** To prevent duplicates and organize the shelves, the system automatically generates location codes in the format F[floor]-[section]-[row]-[bookcode]. When adding copies, the system tracks the highest existing serial number to ensure every copy gets a unique identification number, such as 001 or 002.

- **Quick ID-Based Searching:** The system performs a linear search through the linked list to find specific books by walking from the head pointer. This allows the librarian to instantly pull up a book's status, see who borrowed it, or find its exact shelf location. Since the program checks each book one by one starting from the head, the time complexity for this search is O(n) which is efficient for this system but needs to change later.

- **Automated Fine Calculation:** A specialized module manages the financial side of book returns. By comparing a Unix timestamp (rawDueDate) against the current system time, the system calculates the exact number of overdue days and applies a fine of Rs. 10 per day.

- **Smart Summary and Dashboard:** The dashboard provides a real-time overview of total, borrowed, and available books using a simple list traversal. Additionally, the system group books by their code, showing a summary table of how many copies exist for each title.

4.2 Results and Performance Analysis

This part looks at how well the system works and how fast it handles data.

4.2.1 Functional Outcomes

Testing shows the system works well. The main results are:

- Pointer Reliability: The system was tested by adding and deleting many books. The next and prev pointers always connected correctly. This means the chain of books never broke.
- Logic Correctness: The system handles deleting the first book or the last book correctly. It updates the head and tail pointers so the system can still find the rest of the books.
- Mathematical Accuracy: The fine calculation was checked by hand. The system was 100 percent accurate because it uses the computer clock.

4.2.2 Performance Observations

Because the system is built with C++ and a linked list, it is very fast:

- Fast Insertion: Adding a new book is instant because of the tail pointer. The system does not have to check every book to find the end.
- Resource Efficiency: The program uses very little computer memory. It only uses memory for the books that are actually added.

4.3 Challenges and Limitations

4.3.1 Major Challenges

- The Next Pointer Problem: When deleting many books at once, it was hard to find the next book after one was deleted. I fixed this by saving the address of the next book in a temporary pointer before deleting the current one.
- UI Refreshing: Making the screen update after a book was deleted was tricky. I used a special function to redraw the table every time the list changed.

4.3.2 System Limitations

- Temporary Storage: The books are stored in the computer RAM. This means if you close the program, the books are deleted. A future version would need to save them to a database.
- Large Scale Search: If the library has thousands of books, searching from the start might become a bit slower.

4.3.3 Discussion

The results show that using a doubly linked list is a great way to manage a library. It is much better than using paper or simple spreadsheets. The unique id system was main part. It uses the highest existing number so that IDs are never repeated. Even though it does not save data after closing, it is a very fast and accurate tool.

4.3.4 Summary

The library management system successfully uses data structures to solve library problems. Using pointers makes the system grow easily. The automated fines and location ID’s remove human mistakes. It is a fast and simple tool that shows how digital systems are better for modern libraries.

Chapter 5
Conclusion and Future Works

This chapter concludes the project by summarizing the overall development of the Library Management System and evaluating how the project objectives were achieved. The system successfully turned the theoretical concept of a doubly linked list into a functional desktop application.

The project successfully used a doubly linked list as the main way to store book records. Every important task including adding books at the end, deleting books from any position by relinking pointers, and moving through the list to update the screen was built into a professional interface using the Qt framework.

5.1 Limitations

Even though the main features work well, there are some limitations in the current version:

- **Memory Storage**: All book records are stored in the computer's temporary memory. This means all data is lost when the program is closed because the system does not yet save information to a database.
- **No Login System**: The application does not have a login screen. This means anyone who opens the program can delete books or change student records.
- **Fixed Rules**: Some parts of the system, like the Rs. 10 daily fine and the shelf limits, are locked into the code. They cannot be changed easily through the application .
- **Single User**: The system is made for one person to use at a time. it cannot be shared across a network of multiple computers.

5.2 Future Enhancements

To improve the Library Management System in the future, the following updates are proposed:

- **Permanent Saving**: Adding a way to save book records to database like SQLite so the information stays there even after the computer is turned off.
- **User Accounts**: Creating a secure login system so that only authorized librarians can delete books or manage sensitive student data.

- **Faster Searching**: Using more advanced sorting methods to make searching even faster when the library grows to have thousands of books.
- **Settings Panel**: Adding a settings page where the librarian can change the fine amounts or shelf names without needing to change the computer code.
- **Automatic Reminders**: Adding a feature that automatically marks books as overdue and sends a message or alert when a student owes a fine.

References

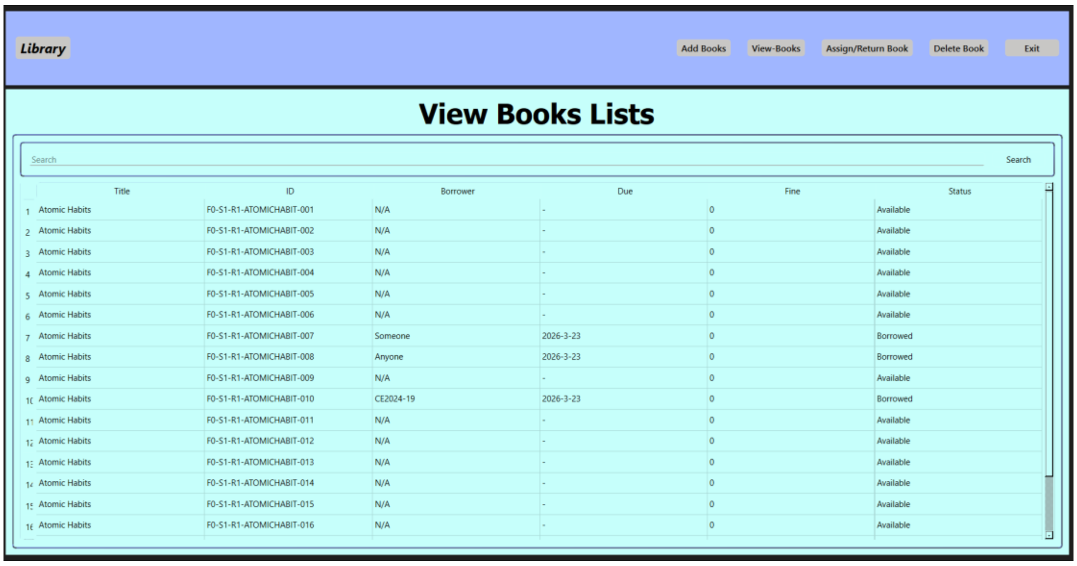
Al-Barazanchi, I. and Abdulshaheed, H. R. (2019). Designing a library management system for gazi husrev-beg library using data structure and algorithm. *Heritage and Sustainable Development*, 1(2):64–71.

Evergreen Project (2024). Evergreen open-source integrated library system (ils). Website. Available at: evergreen-ils.org.

Koha Community (2023). Koha library software. Website. Available at: kohacommunity.org.

Lipschutz, S. (2011). *Data Structures with C*. McGraw-Hill Education.

Appendix



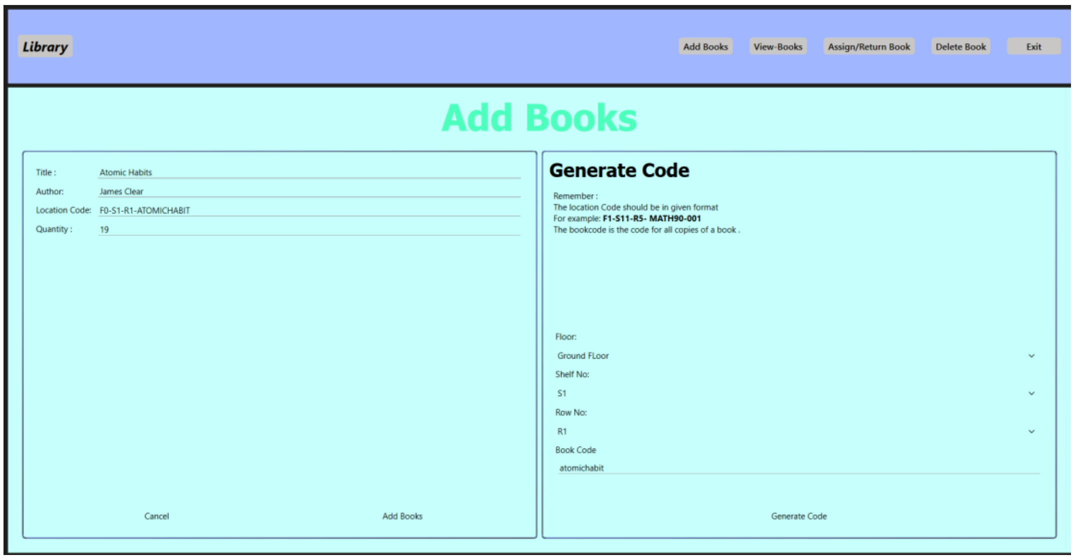
View Books Lists

Search

Search

	Title	ID	Borrower	Due	Fine	Status
1	Atomic Habits	F0-S1-R1-ATOMICHABIT-001	N/A	-	0	Available
2	Atomic Habits	F0-S1-R1-ATOMICHABIT-002	N/A	-	0	Available
3	Atomic Habits	F0-S1-R1-ATOMICHABIT-003	N/A	-	0	Available
4	Atomic Habits	F0-S1-R1-ATOMICHABIT-004	N/A	-	0	Available
5	Atomic Habits	F0-S1-R1-ATOMICHABIT-005	N/A	-	0	Available
6	Atomic Habits	F0-S1-R1-ATOMICHABIT-006	N/A	-	0	Available
7	Atomic Habits	F0-S1-R1-ATOMICHABIT-007	Someone	2026-3-23	0	Borrowed
8	Atomic Habits	F0-S1-R1-ATOMICHABIT-008	Anyone	2026-3-23	0	Borrowed
9	Atomic Habits	F0-S1-R1-ATOMICHABIT-009	N/A	-	0	Available
10	Atomic Habits	F0-S1-R1-ATOMICHABIT-010	CE2024-19	2026-3-23	0	Borrowed
11	Atomic Habits	F0-S1-R1-ATOMICHABIT-011	N/A	-	0	Available
12	Atomic Habits	F0-S1-R1-ATOMICHABIT-012	N/A	-	0	Available
13	Atomic Habits	F0-S1-R1-ATOMICHABIT-013	N/A	-	0	Available
14	Atomic Habits	F0-S1-R1-ATOMICHABIT-014	N/A	-	0	Available
15	Atomic Habits	F0-S1-R1-ATOMICHABIT-015	N/A	-	0	Available
16	Atomic Habits	F0-S1-R1-ATOMICHABIT-016	N/A	-	0	Available

Figure A.1: Books Details Page



Add Books

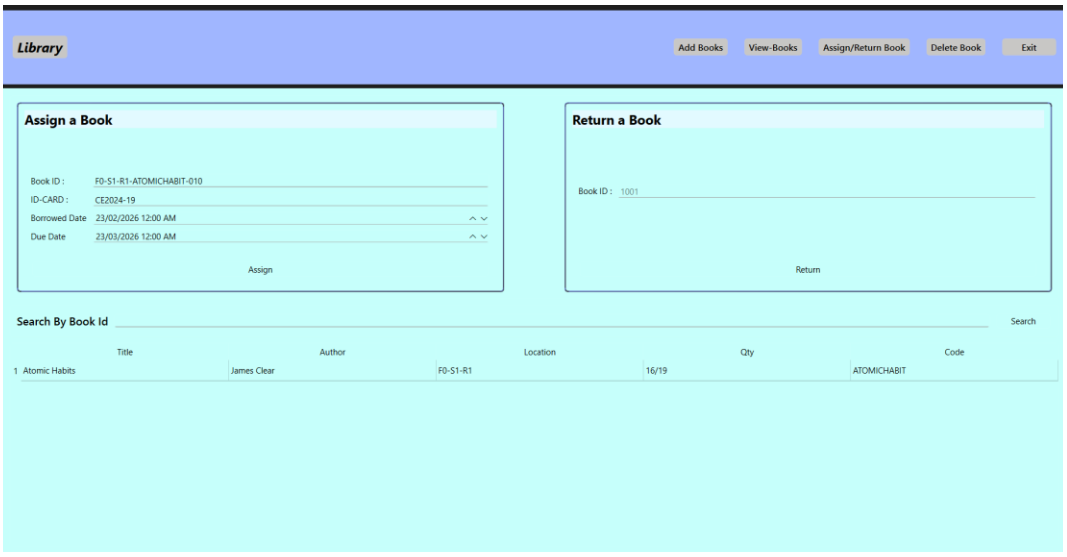
Title : Atomic Habits
Author: James Clear
Location Code: F0-S1-R1-ATOMICHABIT
Quantity : 19

Generate Code

Remember :
The location Code should be in given format
For example: **F1-S11-R5- MATH90-001**
The bookcode is the code for all copies of a book .

Floor: Ground FLoor
Shelf No: S1
Row No: R1
Book Code: atomichabit

Figure A.2: Add Books Page



Assign a Book

Book ID : F0-S1-R1-ATOMICHABIT-010
ID-CARD : CE2024-19
Borrowed Date 23/02/2026 12:00 AM
Due Date 23/03/2026 12:00 AM

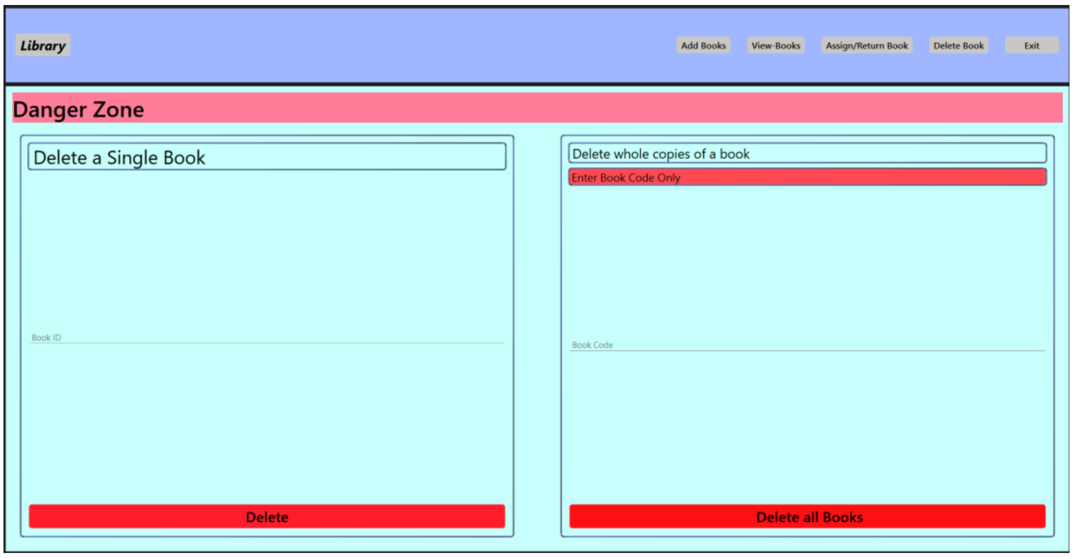
Return a Book

Book ID : 1001

Search By Book Id

	Title	Author	Location	Qty	Code
1	Atomic Habits	James Clear	F0-S1-R1	16/19	ATOMICHABIT

Figure A.3: Assign/Return Page



Danger Zone

Delete a Single Book

Book ID

Delete whole copies of a book

Enter Book Code Only

Book Code

Figure A.4: Delete Books Page