2025S_CSCI441_VB_Software Engineering

**SOFTWARE ENGINEERING PROJECT**

**#Report #2 - PART 2 : *Interaction Diagrams* and Section, *Class Diagram and Interface Specification*)**

Digitized Library Management System

**Team C**

Antony Reyes Pilo,  Channet Lor, David Harmon, Matthew Herrick, Viraksith Huor

Instructor: Dr. Mike Mireku Kwakye

Feb 16th, 2025

Fort Hays State University

| Individual Contributions Breakdown table | | | | | |
|---|---|---|---|---|---|
| **Task** | **Antony Reyes Pilo,** | **Channet Lor** | **David Harmon** | **Matthew Herrick** | **Viraksith Huor** |
| **Report 1 - Part 1** | | | | | |
| **Cover Page** | | X | | | |
| **Summary of Changes** | x | | | x | |
| **Individual Contributions Breakdown** | x | x | x | x | x |
| **1. Customer Problem Statement**<br>**a. Problem Statement** | x | | | x | |
| **b. Decomposition into Sub-Problems**<br>**c. Glossary of Terms** | | | x | | |
| **2. Goals, Requirements and Analysis**<br>**2.a Business Goals** | | x | | | |
| **2.d User Interface Requirements**<br>**2.b Functional Requirements**<br>**2.c Non-Functional Requirements** | | | | | x |
| **Report 1 - Part 2** | | | | | |
| **3. Use Cases**<br>**a. Stakeholders**<br>**b. Actors and Goals** | | | | | x |
| **c. Use Cases**<br>**i. Casual Description**<br>**ii. Use Case Diagram**<br>**iii. Traceability Matrix** | | | | x | |
| **iv. Fully Dressed Description** | | | x | | |
| **d. System Sequence Diagrams**<br>**i. Login**<br>**ii. Reservations** | x | | | | |
| **iii. Database**<br>**iv. Admin Dashboard** | x | | | | |
| **4. User Interface Specification**<br>**a. Preliminary Design** | | X | | | |

| | | | | | |
|---|---|---|---|---|---|
| **b. User Effort Estimation** | | | | | |
| **Report #1: Full Report** | | | | | |
| **5. System Architecture and System Design**<br>**a. Identifying Subsystems** | X | | | | |
| **b. Architecture Styles** | | | X | | |
| **c. Mapping Subsystems to Hardware** | | X | | | |
| **d. Connectors and Network Protocols** | | | | X | |
| **e. Global Control Flow** | | X | | | |
| **f. Hardware Requirements** | | | | | X |
| **Report #2: Part 1** | | | | | |
| **6. Domain Analysis**<br>**a. Conceptual model** | | X | | | |
| **I. Concepts Definition** | | X | | | |
| **II. Association definitions** | | | X | | |
| **III. Attribute Definitions** | X | | | | |
| **IV. Traceability Matrix** | | | X | | |
| **b. System Operation Contracts** | | | | | X |
| **c. Data Model and Persistent Data Storage** | | | | X | |
| **Report #2: Part 2** | | | | | |
| **7. Interaction Diagrams**<br>**a. UC-1: SearchBooks Diagram** | X | | | X | |
| **b.UC-2: CheckBookAvailability Diagram** | X | | | X | |
| **c. UC-3: ManageBookCheckout Diagram** | X | | | X | |
| **d. UC-4: AccessHelpSection Diagram** | X | | | X | |

| | | | | | |
|---|---|---|---|---|---|
| **e.UC-5: ManageUserAccount diagram** | X | | X | X | |
| **f. UC-6: ReserveBook diagram** | X | | X | X | |
| **g. UC-7: ManageLibraryInventory diagram** | | | | X | |
| **8. Class Diagram and Interface Specification** | | X | | | |
| **a. Class Diagram** | | X | | | |
| **b. Data Types and Operation Signatures** | | X | | | |
| **c. Design Patterns** | | X | | | |
| **d. Object Constraint Language (OCL)** | | | | | X |
| **Report #2: Full report** | | | | | |
| **9. System Architecture and System Design**<br>**a. Identifying Subsystems**<br>**b. Architecture Styles** | | | | | |
| **c. Mapping Subsystem to Hardware**<br>**d. Connectors and Network Protocols** | | | | | |
| **e. Global Control Flow**<br>**f. Hardware Requirements** | | | | | |
| **10. Algorithms and Data Structures**<br>**a. Data Structures**<br>**b. Concurrency** | | | | | |
| **11. User Interface Design and Implementation** | | | | | |
| **12. Design of Tests**<br>**a. Test Cases**<br>**b. Test Coverage** | | | | | |
| **c. Solution**<br>**d. Conclusion** | | | | | |
| **13. History of Work, Current Status, and Future Work**<br>**a. Merging the Contributions from Individual Team Members** | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| **b. Project Coordination and Progress Report** | | | | | |
| **14. Reflective Essay and Peer Evaluation** | | | | | |
| **Plan of work** | | | | X | |
| **Project Management** | | X | | | |
| **References** | X | X | X | X | X |

# Table of Contents

**Summary of Changes**

- Included “Interaction Diagrams” and explained their design principles
- Included “Class Diagram and Interface Specification”

**Work Assignment**

- Antony Reyes Pilo: HTML, Python, SQL, CSS, C++, Java, JavaScript,, PHP.
    - Implementation of Special Features(e.g., search filters)
    - Report Editor
    - Front-end Software
    - UI Developer
- Channet Lor: HTML, CSS, JS, Python, SQL,
    - Report Editor
    - Project Management
    - Reference Collector
- David Harmon: Java/JavaScript, PHP, SQL, CSS, Git
    - Full-Stack Development Support
    - Implementation of Special Features(e.g., search filters)
    - Version Control & Automation Scripts
- Matthew Herrick (Team Lead): SQL, JavaScript, HTML
    - Database Management
    - Front-end Software
    - Report Editor
- Viraksith Huor: HTML, CSS, JavaScript, Python, C++, SQL
    - UX UI Design
    - Report Editor

==========

Report #1 - Part 1

**1. Customer Problem Statement**

## a. Problem Statement:

As a lifelong lover of books and a frequent patron of my local library, I have always cherished the quiet aisles filled with endless possibilities. However, as much as I adore the traditional library experience, I cannot ignore the growing frustrations I face as a user in this increasingly digital world. Libraries, as vital community institutions, are struggling to keep up with the pace of technological advancement, and this is creating a disconnect between what libraries offer and what patrons like me need. The current system feels outdated, cumbersome, and inefficient, and it is high time for a transformation. A digitized library management system could be the solution to these challenges, and I believe it would revolutionize the way libraries operate and serve their patrons.

**The Problem: Outdated Systems and Frustrating Experiences**

The primary issue I face as a library patron is the inefficiency of the current system. Searching for a book is often a time-consuming and frustrating process. The catalog system is usually outdated, requiring me to scroll through endless lists or manually search through physical index cards. Even when I do find a book I want, there is no guarantee it will be available. I often have to visit the library multiple times to check if a book has been returned, only to find out it is still checked out. This lack of real-time information wastes my time and diminishes my enthusiasm for using the library. Another significant problem is the difficulty I have in managing my account and keeping track of my borrowed books. I often forget when my books are due, leading to late fees that could have easily been avoided with better reminders. Additionally, I wish I could create a wishlist of books I want to read or easily revisit my borrowing history to recall titles I enjoyed. These features are standard in online retail platforms, yet they are absent in most library systems.

For librarians, the challenges are equally frustrating. Adding or removing books from the catalog seems like a manual and tedious process. Managing user accounts, tracking overdue books, and handling reservations are all tasks that could be streamlined with the right tools. Librarians are incredibly knowledgeable and helpful, but they can often be bogged down by administrative tasks that take away from their ability to assist patrons like me.

**How a Digitized Library Management System Could Help**

A Digitized Library Management System has the potential to address these pain points and transform the library experience for both patrons and librarians. Here's how I envision such a system improving my experience:

***1. Effortless Search and Discovery***

The system should allow me to search for books using multiple filters, such as title, author, genre, publishing date, and even reading level. This would make it easy to either find exactly what I'm looking for or discover new books that match my interests. A user-friendly interface with real-time availability information would save me trips to the library and tedious phone calls and help me plan my visits more effectively.

***2. Real-Time Availability and Reservations***

One of the most frustrating aspects of the current system is not knowing whether a book is available. A digitized system should display real-time availability and allow me to reserve books in advance. If a book is checked out, I should be able to join a waiting list and receive notifications when it becomes available. This would eliminate the guesswork and ensure I never miss out on a book I want to read.

***3. Personalized User Accounts***

Each user should have a personalized account that includes a wishlist, borrowing history, and due date reminders. This would help me keep track of the books I've read, plan my future reading, and avoid late fees. The system could also recommend books based on my reading history, making it easier to discover new titles I might enjoy.

***4. Streamlined Check-In and Check-Out Process***

Checking books in and out should be a seamless process, whether I'm doing it online or in person. The system should allow me to manage my borrowed books from my account, renew them if necessary, and receive reminders when they are due. This would make the entire process more convenient and reduce the stress of managing due dates.

***5. Enhanced Communication and Support***

A built-in help section with FAQs and the ability to ask a librarian questions would be incredibly useful. Whether I need help finding a book or have a question about my account, having access to support directly through the system would save time and improve my overall experience.

***6. Simplified Management for Librarians***

Librarians should have an intuitive interface for managing the library's catalog, user accounts, and reservations. Adding or removing books should be as simple as a few clicks, and the system should automate tasks like tracking overdue books and sending reminders. This would free up librarians to focus on what they do best—helping patrons and fostering a love of reading.

**The Impact of a Digitized Library Management System**

Implementing a Digitized Library Management System would not only address the frustrations I face as a patron but also modernize libraries as a whole. It would make libraries more accessible and appealing to a wider audience, including younger generations who are accustomed to the convenience of digital platforms. By streamlining operations and enhancing the user experience, this system would ensure that libraries remain relevant and valuable in the digital age. For me, this system would mean less time spent searching and more time spent reading. It would mean fewer late fees, more personalized recommendations, and a smoother overall experience. For librarians, it would mean less time spent on administrative tasks and more time engaging with the community. And for libraries, it would mean staying competitive in a world where digital convenience is no longer a luxury but an expectation.

In conclusion, the need for a Digitized Library Management System is clear. It is a solution that benefits everyone—patrons, librarians, and the libraries themselves. By embracing this technology, libraries can continue to be a cornerstone of our communities, offering knowledge, inspiration, and connection in a way that meets the needs of the modern world. I eagerly await the day when I can walk into my local library and experience the seamless, efficient, and user-friendly system that this project promises to deliver.

## b. Decomposition into Sub-Problems

To develop a more efficient and user-friendly Digitized Library Management System, we need to break down the problem into specific areas that need improvement. Addressing each of these ensures smoother operations for both library staff and users.

1. **Search & Filtering**
   Users should have a fast and flexible system to find books using:

   - Title, Author, Genre, Reading Level, ISBN
   - Availability status (Checked out, Available, Reserved, etc.)
   - Smart search enhancements such as autocomplete and filtering

2. **Book Borrowing & Return System**

   - The system should automate check-ins and check-outs to eliminate manual errors.
   - Borrowers should receive real-time updates on due dates and book availability.
   - Late returns should trigger automated overdue reminders to users.

3. **User Accounts & Borrowing History**
   Users should be able to log into an account to:

   - View current checkouts
   - See due dates and overdue notices
   - Renew books if possible
   - Reserve books in advance when they become available
   - Access their borrowing history to track past reads

4. **Library Inventory Management**
   Librarians need an easy-to-use admin panel to:

   - Add new books
   - Remove outdated books
   - Update book availability status (damaged, missing, checked out, etc.)
   - Manage waitlists for popular books

   The system should support bulk uploads of book data for faster catalog updates.

5. **Notifications & Communication**
   The system should send automated notifications for:

   - Book due dates & overdue alerts
   - Reservation confirmations & availability updates
   - Account-related reminders (password resets, expiring holds, and borrowing limits)

Users should be able to ask questions to library staff directly through a Help/Support feature.

## c. Glossary of Terms

- **Library Account** – A personal account that allows users to log in, search for books, check out items, and manage their borrowing history.
- **Catalog Search** – A feature that enables users to look up books, e-books, audiobooks, and other library materials using keywords, author names, or ISBNs.
- **Check-Out** – The process of borrowing a book or other material from the library for a specified period.
- **Due Date** – The deadline by which a borrowed item must be returned to avoid late fees or penalties.
- **Hold (**king check-outs.
- **E-Book** – A digital version of a book that users can read online or download.
- **Audiobook** – A recorded version of a book that users can listen to online or download.
- **Reading History** – A log of books previously checked out by the user, available for reference.
- **Book Recommendation System** – A feature that suggests books to users based on their reading history and preferences.
- **Interlibrary Loan (ILL)** – A service that allows users to request books from other libraries if they are not available in their local library.
- **Self-Checkout** – A feature that allows users to check out books themselves using a kiosk or a mobile app.
- **Fines & Fees** – Any charges applied to a user's account due to late returns, lost books, or damaged materials.
- **Library Hours** – The operating hours during which users can visit the physical library location.
- **Digital Collection** – A collection of online materials, including e-books, audiobooks, and research databases, available through the library website.
- **User Dashboard** – A personalized section of the library website where users can view their checked-out books, holds, due dates, and fines.

## 2. Goals, Requirements, and Analysis

### a. Business Goals


Business Goals Canva Link

The 5 point in details:

- **Automate Library Operations:** Streamlines book cataloging, borrowing, and returns. Reduces manual work and improves efficiency.
- **Enhance User Accessibility:** Allows users to search, reserve, and renew books online. Provides a seamless and convenient experience.
- **Improve Inventory Management:** Tracks book availability, issuance, and overdue returns. Helps prevent book loss and misplacement.
- **Ensure Data Security:** Implements authentication and role-based access. Protects user data and library records from unauthorized access.
- **Integrate Digital Resources:** Provides access to e-books, research papers, and online journals. Expands learning opportunities beyond physical books.

## b. Enumerated Functional Requirements

| Identifier | PW | Requirement |
|---|---|---|
| REQ1a | 5 | Users should be able to search for books using titles |
| REQ1b | 5 | Users should be able to search for books using author name. |
| REQ1c | 5 | Users should be able to search for books and filter by reading level |

| | | |
|---|---|---|
| REQ1d | 5 | Users should be able to search for books using ISBN |
| REQ1e | 5 | Users should be able to search for books and filter by publishing date |
| REQ2 | 4 | The system should track and display the real-time availability for each book. |
| REQ3a | 3 | Users should be able to check in books. |
| REQ3b | 3 | Users should be able to check out books. |
| REQ4a | 2 | The system should have a help section to request assistance. |
| REQ4b | 2 | The help section should provide to send questions to a librarian |
| REQ4c | 2 | The help section should have a list of frequently asked questions (FAQ) |
| REQ5a | 4 | Users should be able to create an account |
| REQ5b | 3 | Each account should have a borrowing history |
| REQ5c | 4 | Each account should have a wishlist |
| REQ6 | 4 | The system should allow users to reserve books in advance when become available. |
| REQ7a | 5 | Librarians should have a simple way to add books |
| REQ7b | 5 | Librarians should have a simple way to remove books |

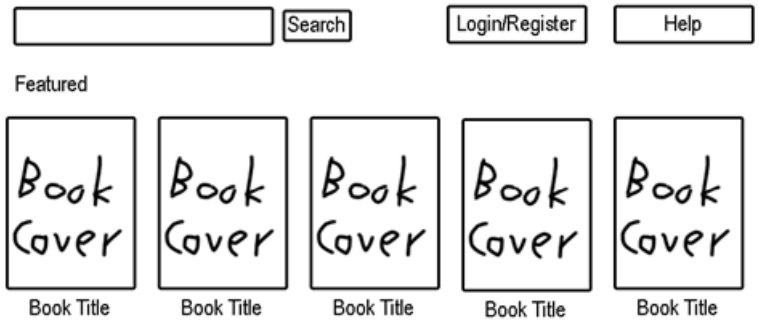
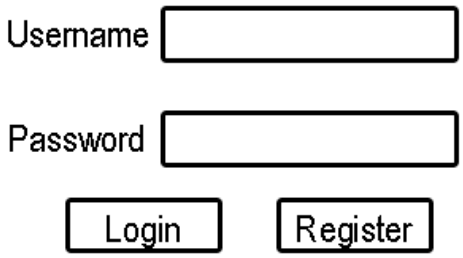
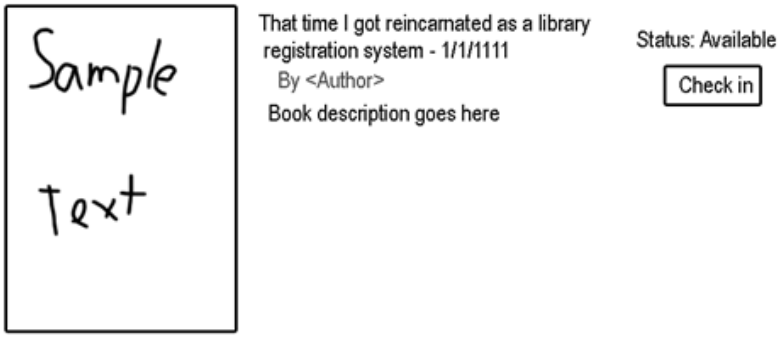
c. Enumerated Nonfunctional Requirements

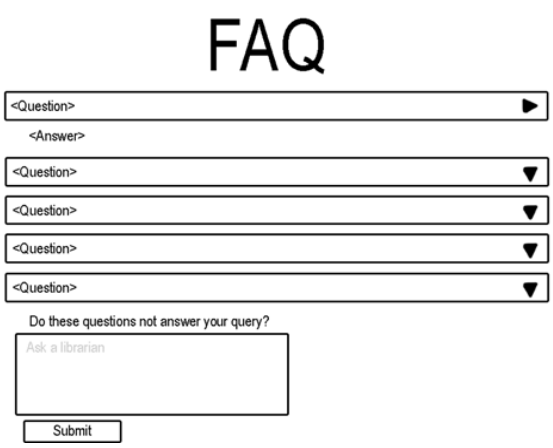
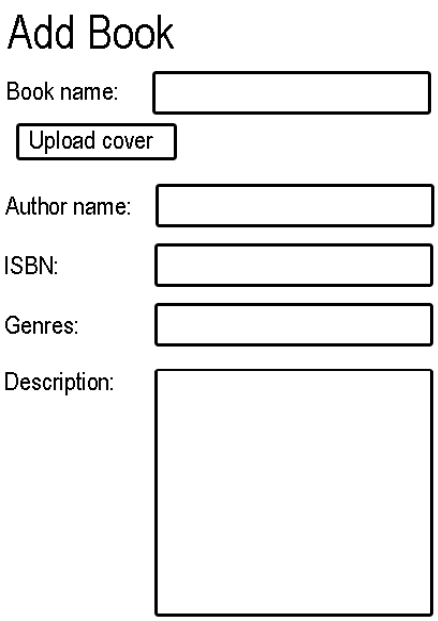
| Identifier | PW | Requirement |
|---|---|---|
| REQ8 | 5 | The system should run reliably with no downtime |
| REQ9 | 4 | The interface should be clear and easy to use, avoiding technical language. |
| REQ10 | 5 | The user data must be securely handled to protect privacy |
| REQ11 | 5 | The system should synchronize information seamlessly for book status (available, unavailable and waiting list) |
| REQ12a | 5 | Librarians should always have access information about book status (available, unavailable and waiting list) |
| REQ13 | 3 | The system should be scalable to accommodate libraries of different sizes |

**FURPS Table:**

| | |
|---|---|
| Functionality | • Features a home page that displays books, has a login/register button, search bar and a help button<br>• The login/register button takes the user to a page that allows them to either input their account details for registration or login<br>• The search bar is used to narrow down the list of books<br>• Clicking a book takes the user to a separate page with the book's details.<br>• On the book's page, the user can check in the book if the book is available.<br>• A help button takes the user to a separate page that has the FAQ and provides the user a method to contact library staff<br>• Power users like librarians have a separate page to add new books |
| Usability | • The interface should be clear and easy to use<br>• The interface will avoid using technical language<br>• Input fields will only accept valid types |
| Reliability | • The system will have minimal issues<br>• Any downtime that occurs will be a result of external factors such as the server it is hosted on |
| Performance | • The system will process requests and retrieve information in a timely manner |
| Supportability | • The system will be supported on multiple mainstream platforms |

d. User Interface Requirements

| Identifier | PW | Requirement |
|---|---|---|
| REQ14 | 5 | Homepage mockup<br>  |
| REQ15 | 5 | Login/register page mockup<br>  |
| REQ16 | 5 | Book screen mockup<br>  |

| | | |
|---|---|---|
| REQ17 | 4 | FAQ page mockup<br>  |
| REQ18 | 5 | Librarian's book adder page mockup<br>  |

==========

Report #1 - Part 2

## 3. Functional Requirement Specification and Use Cases

### a. Stakeholders

1. **Library Patrons:** Individuals who borrow books, reserve items, and use library services. Their main interests include easy book search, real-time availability, account management, and due date notification.

2. **Librarians:** Staff responsible for managing the catalog, assisting patrons, and overseeing library operations. They need efficient book management, automated overdue reminders, and streamlined account management.

3. **System Administrators:** The team responsible for keeping the system running smoothly and ensuring it is available for users at all times. They handle any issues that might cause the system to stop working and make sure the system is secure and protected from unauthorized access.

4. **Library Management:** Library administrators who make decisions on the library's services. They aim to ensure the system supports the library's mission, budget, and service goals.

5. **External Service Providers:** Third-party vendors and services, such as e-book providers and payment systems, that integrate with the library system.

6. **Regulatory Bodies:** Entities ensuring compliance with legal regulations, especially related to data protection and intellectual property.

7. **Development Team:** The team responsible for building and maintaining the system, ensuring it meets all technical and functional requirements.

### b. Actors and Goals

1. **Library Patron**

   **Type:** Initiating Actor

   **Goal:** To search for books, reserve items, borrow books, and manage their personal account (including viewing due dates and borrowing history.)

2. **Librarian**

   **Type:** Initiating Actor

   **Goal:** To efficiently manage the library catalog, process check-ins and check-outs, and assist patrons with their needs.

3. **System Administrator**

   **Type:** Participating Actor

   **Goal:** To ensure the system runs smoothly and remains available at all times by promptly resolving any issues and maintaining security.

4. **External Service Provider**

   **Type:** Participating Actor

   **Goal:** To seamlessly integrate digital services (such as e-book access or payment processing) with the library system.

5. **Library Management**

   **Type:** Participating/Initiating Actor

   **Goal:** To oversee system performance and ensure that the system supports the library's mission, budget, and service goals.

6. **Development Team**

   **Type:** Participating Actor

   **Goal:** To design, develop, and continuously improve the system so that it meets all functional and non-functional requirements.

**c. Use Cases**

i. **Casual Description**

---

**UC-1: SearchBooks** — Allows Users to search for books using various criteria.
- *Extension point*: Users can search by title.
- *Extension point*: Users can search by author name.
- *Extension point*: Users can filter search results by reading level.
- *Extension point*: Users can search by ISBN.
- *Extension point*: Users can filter search results by publishing date.
- *Derived from requirements*: REQ1a - REQ1e

---

**UC-2: CheckBookAvailability** — Allows Users to view the real-time availability status of a book (optional sub use case, «extend» UC-1: SearchBooks)
- *Derived from requirement:* REQ2

---

**UC-3: ManageBookCheckout** — Allows Users to check out and check in books (mandatory sub use case «include» from UC-5: ManageUserAccount)
- *Extension point*: Users can check out books.
- *Extension point*: Users can check in books.
- *Derived from requirements*: REQ3a, REQ3b

---

**UC-4: AccessHelpSection** — Provides Users with assistance resources.
- *Extension point*: Users can view a list of frequently asked questions (FAQ).
- *Extension point*: Users can send questions to a librarian.
- *Derived from requirements*: REQ4a - REQ4c

---

**UC-5: ManageUserAccount** — Allows Users to create and manage their accounts.
- *Extension point*: Users can create an account.
- *Extension point*: Users can view their borrowing history.
- *Extension point*: Users can maintain a wishlist.
- *Derived from requirements*: REQ5a - REQ5c

---

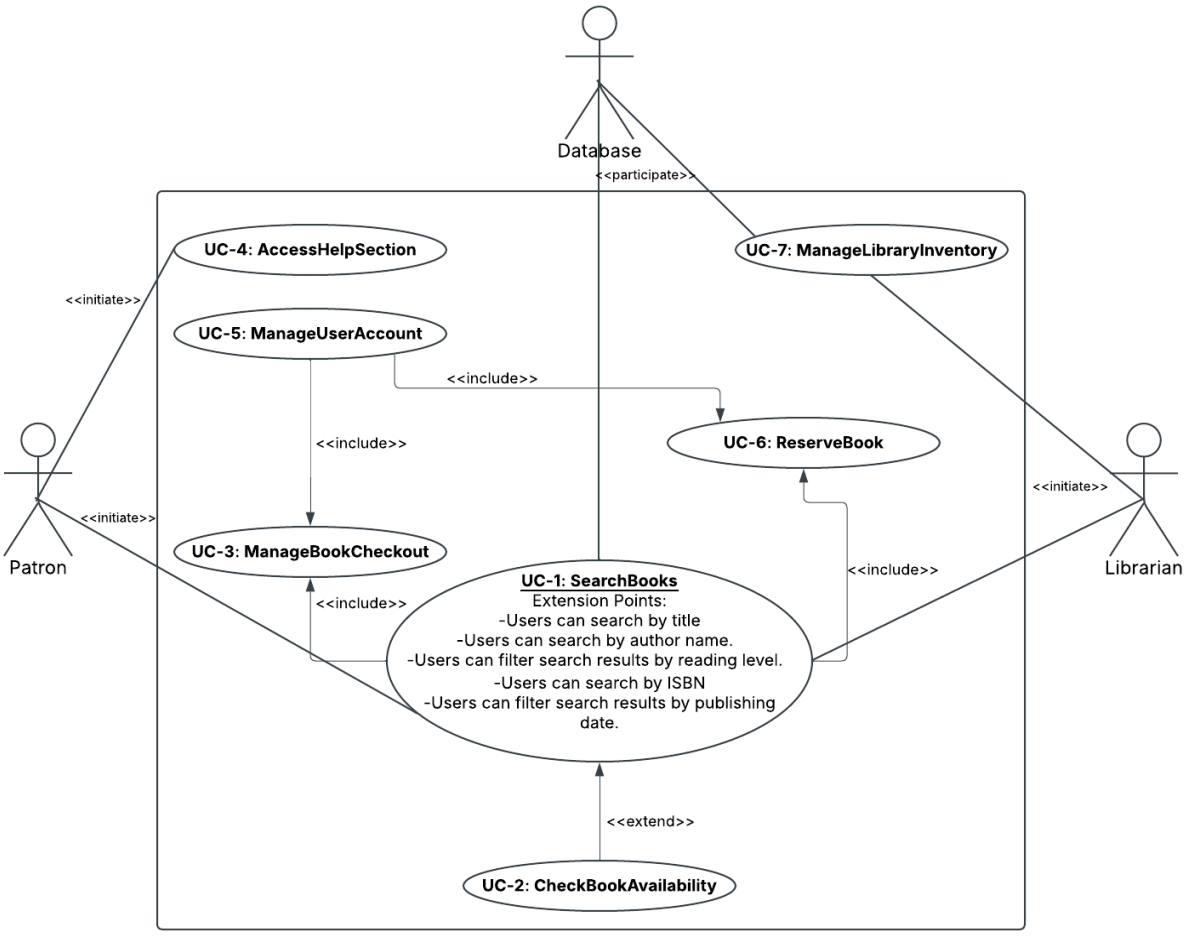
**UC-6: ReserveBook** — Allows Users to reserve books in advance when they become available (mandatory sub use case «include» from UC-5: ManageUserAccount).
- *Derived from requirement*: REQ6

---

**UC-7: ManageLibraryInventory** — Allows Librarians to add or remove books from the library system.
- *Extension point*: Librarians can add books.
- *Extension point*: Librarians can remove books..
- *Derived from requirements*: REQ7a, REQ7b

---

ii. User Case Diagram




**System: Digitized Library Management System**

iii. Traceability Matrix

| Req't | PW | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 |
|---|---|---|---|---|---|---|---|---|
| **REQ1a** | 5 | X | X | | | | | |
| **REQ1b** | 5 | X | X | | | | | |
| **REQ1c** | 5 | X | X | | | | | |
| **REQ1d** | 5 | X | X | | | | | |
| **REQ1e** | 5 | X | X | | | | | |
| **REQ2** | 4 | | X | | | | | |
| **REQ3a** | 3 | | | X | | | | |
| **REQ3b** | 3 | | | X | | | | |
| **REQ4a** | 2 | | | | X | | | |
| **REQ4b** | 2 | | | | X | | | |
| **REQ4c** | 2 | | | | X | | | |
| **REQ5a** | 4 | | | X | | X | X | |
| **REQ5b** | 3 | | | X | | X | X | |
| **REQ5c** | 4 | | | X | | X | X | |
| **REQ6** | 4 | | | | | | X | |
| **REQ7a** | 5 | | | | | | | X |
| **REQ7b** | 5 | | | | | | | X |
| **Max PW** | | 5 | 5 | 4 | 2 | 4 | 4 | 5 |
| **Total PW** | | 20 | 24 | 17 | 6 | 11 | 15 | 10 |

iv. Fully-Dressed Description

**Fully-Dressed Use Case: SearchBooks**
**Use Case Name:** SearchBooks
**Priority:** High
**Primary Actor:** Library Patron(User)
**Goal:** Allow the user to find books using various search criteria (title, author, ISBN, reading level, etc.)
**Scope & Level:** System scope; user-goal level

**Preconditions:**
The user is on the homepage or dashboard and has access to the search functionality.
The library catalog is current and available in the system database.

**Postconditions:**
The system displays a list of books that match the search criteria, including key details like title, author, and availability.
The user can select a book for more details or refine their search as needed.

**Main Success Scenario (Step-by-Step):**

1. The user navigates to the search page via the homepage or dashboard.
2. The system displays the search interface with fields for title, author, ISBN, and optional filters (e.g., reading level, publishing date).
3. The user enters a search term (e.g., author's name) and clicks the "Search" button.
4. The system processes the query by querying the library catalog database.
5. The system presents a list of matching books with basic details for each entry.
6. The user reviews the list and selects a book to view further details, or refines the search if needed.

**Extensions (Alternate Flows):**

- **Invalid Input:** If the user enters nonsensical or invalid characters, the system displays an error message such as "Invalid input. Please try again."

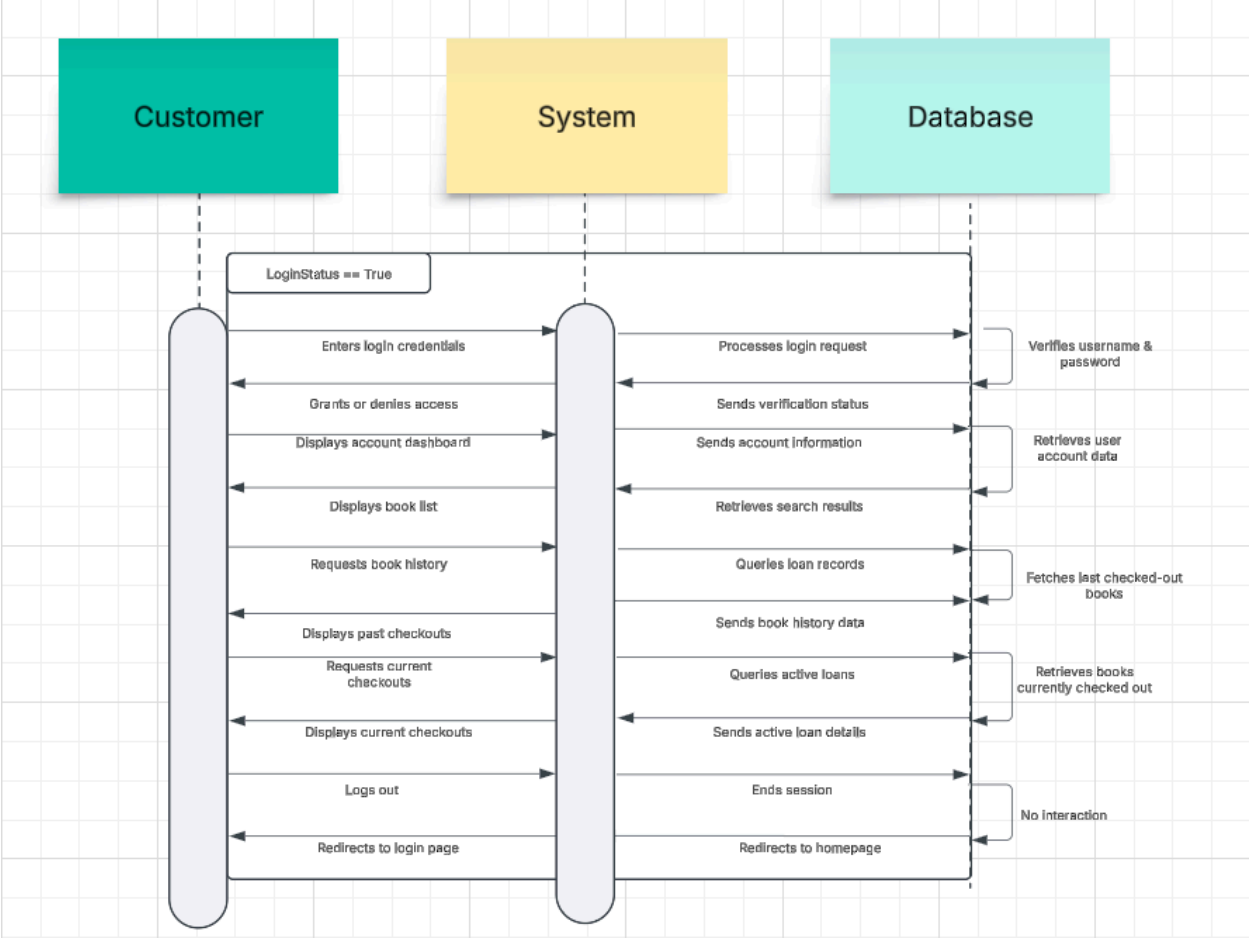
- **No Matches:** If no books match the search criteria, the system displays "No results found" and suggests adjusting the search terms of filters.

**Related Requirements:**

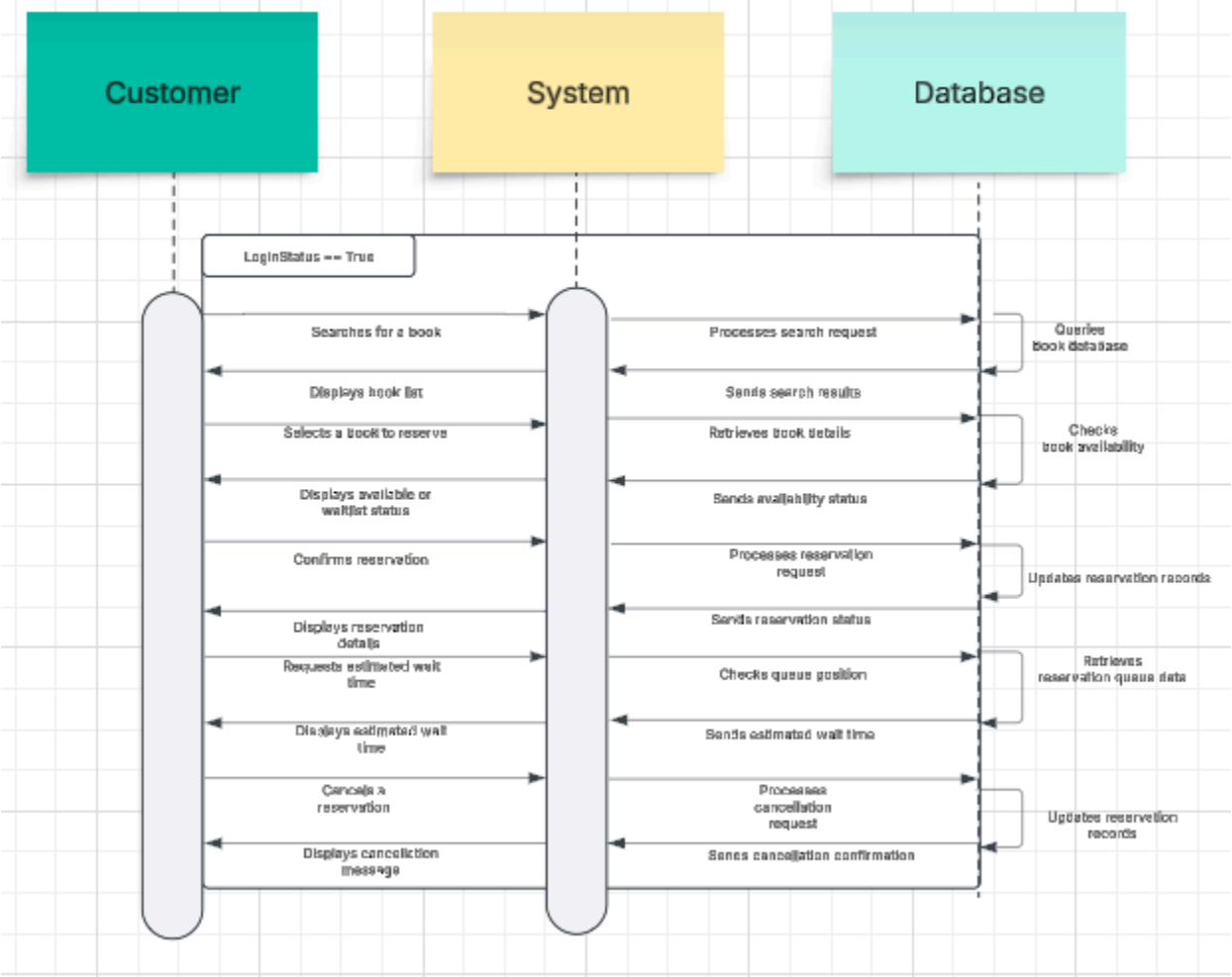
- REQ1a (Search by title)
- REQ1b (Search by author)
- REQ1c (Filter by reading level)
- REQ1d (Search by ISBN)
- REQ1e (Filter by publishing date)
- REQ2 (Display real-time availability)

## d. System Sequence Diagrams

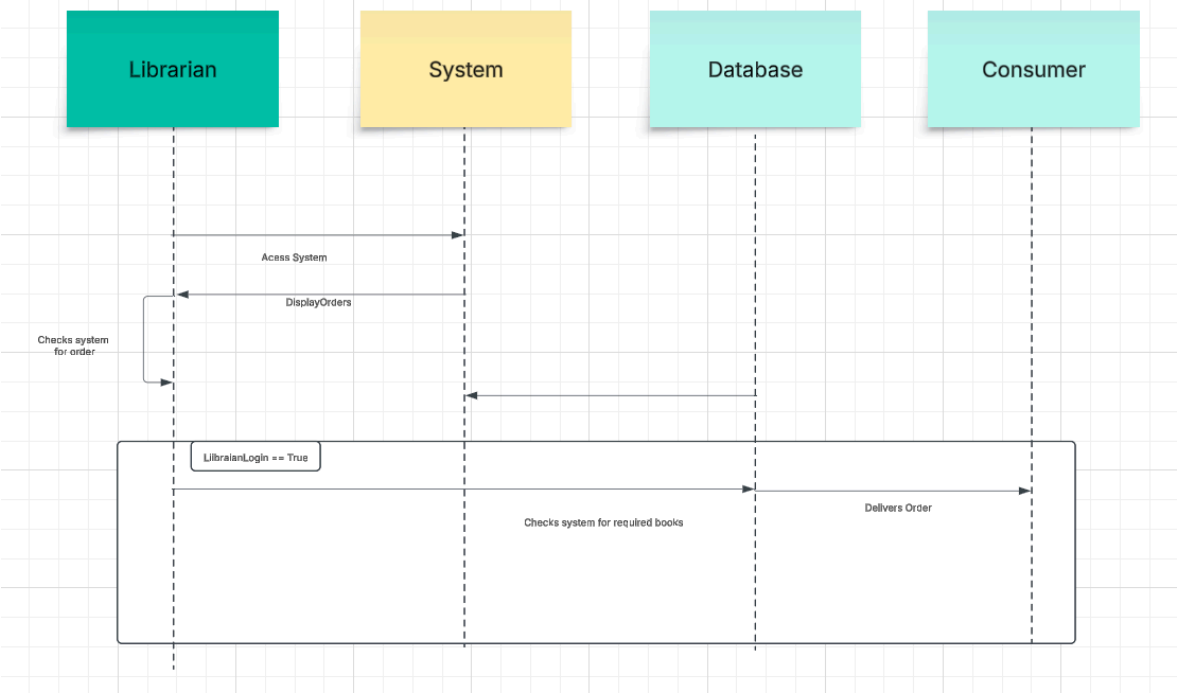
- **Login (update)**



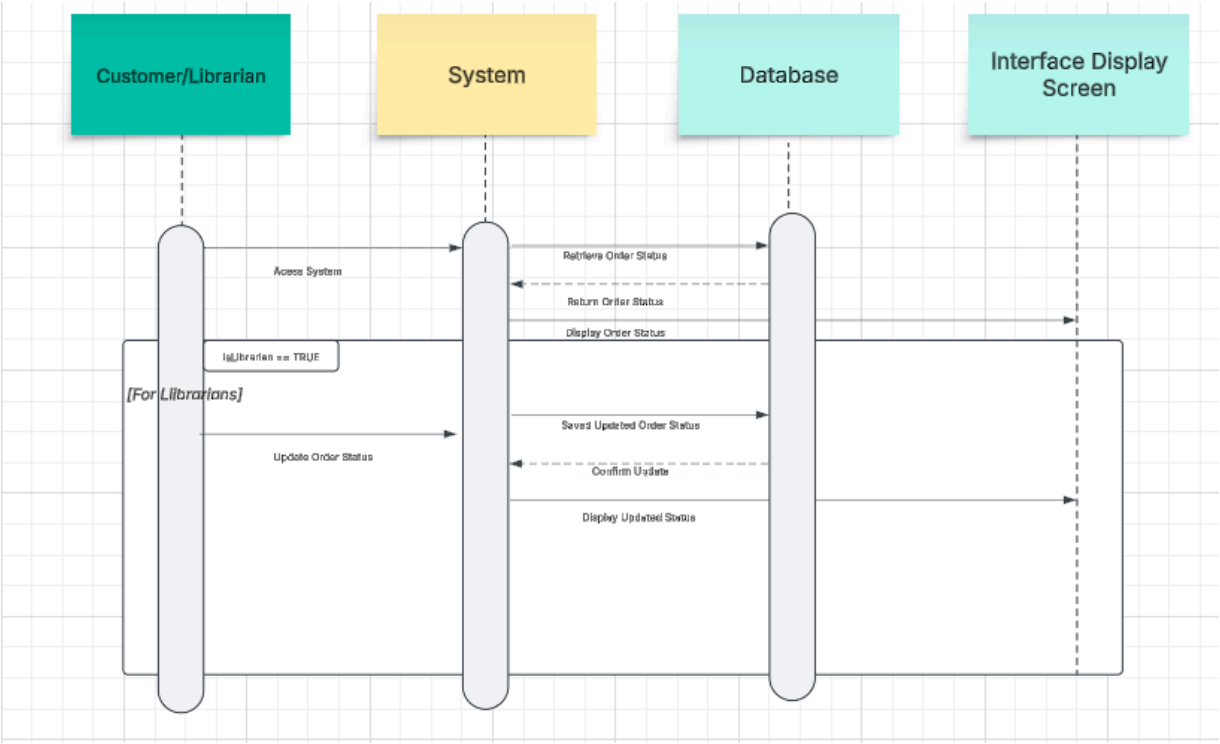
- **Reservations (update)**



- **Database (update)**

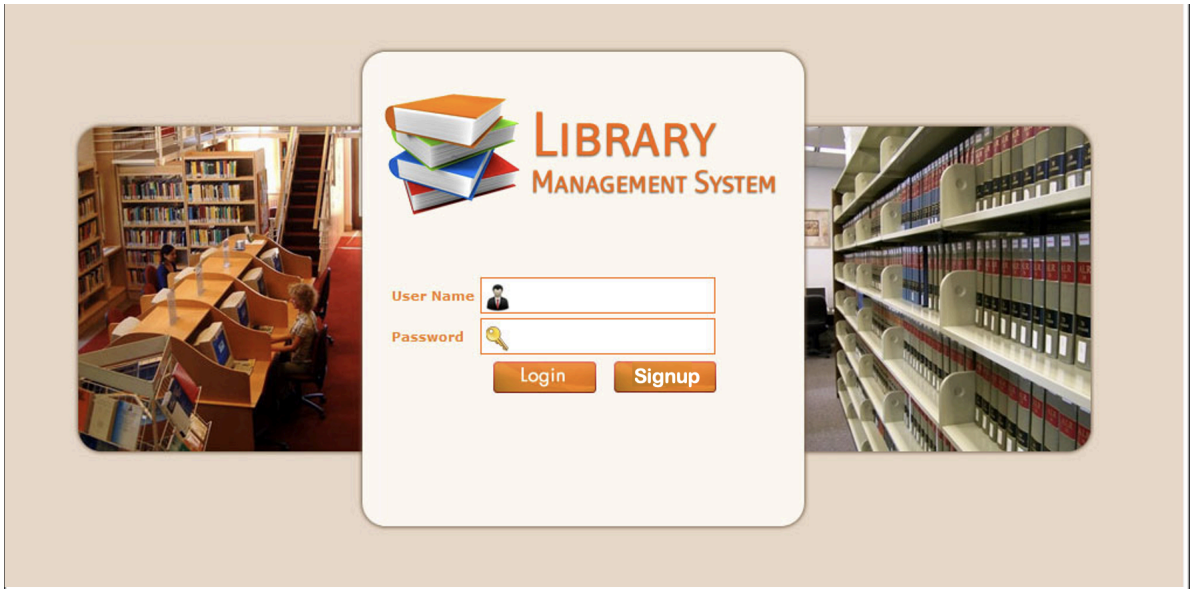



- **Librarian Dashboard (update)**

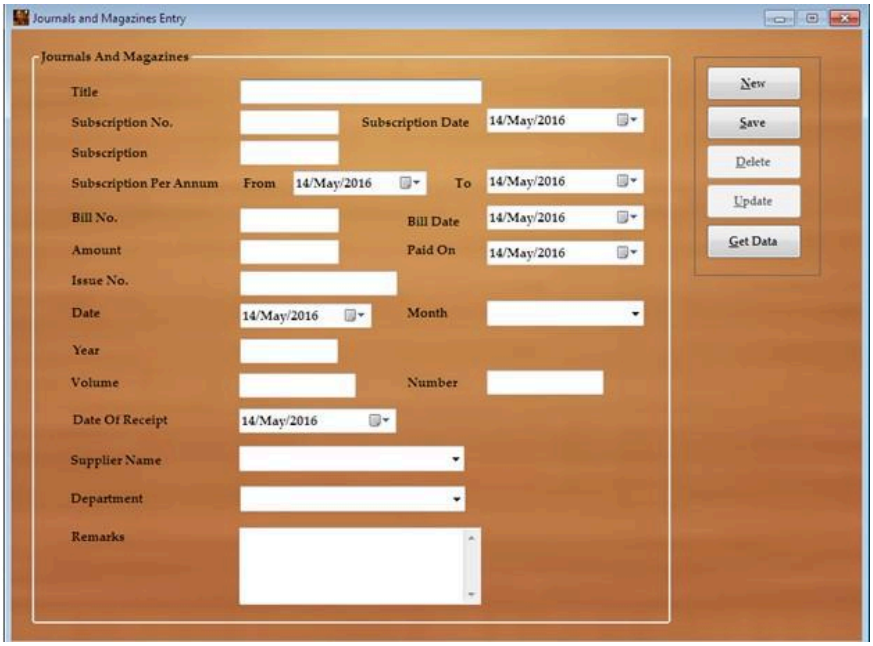


## 4. User Interface Specification

**Example of User Interfaces (UI) :**
**Login or Signup page :**

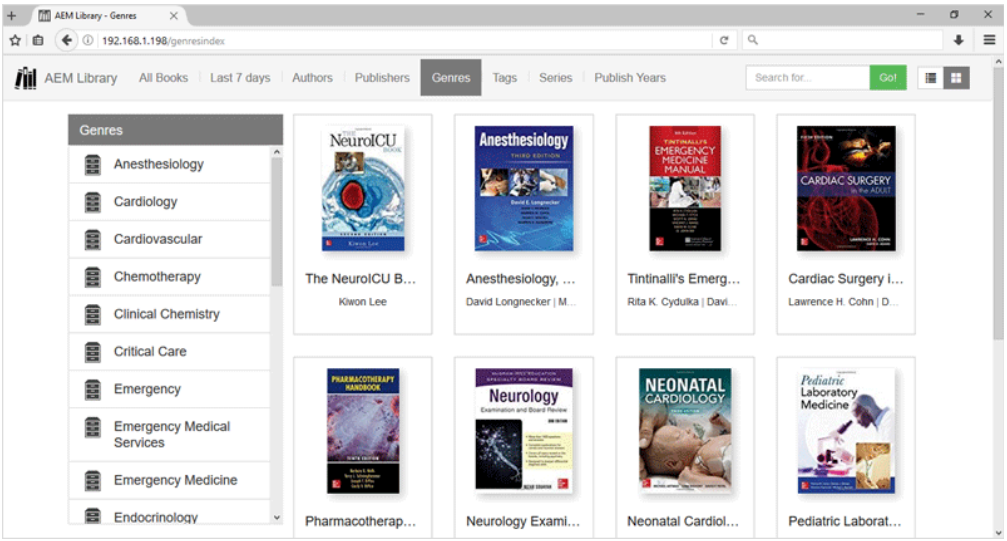



**Signup page:**




[Link to the UI]

**Home page:**




[Home page link]

**a. Preliminary Design**

Preliminary Design for Digitized Library Management System

**Use Case 1: User Registration and Login**

**Step-by-Step Process:**

1. **Start Point:**
   The user visits the Homepage and sees options like Register, Login, Search for Books, and View Borrowed Books.

2. **User Action:**
   The user clicks the Register button to create a new account.

3. **System Response**:
   The system displays the Registration Form, asking for:

   - Full Name
   - Username
   - Email Address
   - Password and Confirm Password

4. **User Action:**
   The user fills in the form and clicks Submit.

5. **System Response:**

   - If the form is valid, the system shows: "Registration successful! Please log in."
   - If there's an error (e.g., invalid email, mismatched password), the system displays an error message.

6. **Navigational Path:**

   Homepage → Register → Registration Form → Confirmation Page.

**Display Mock-up**:

- Registration Form: Fields: Full Name, Username, Email, Password, Confirm Password.
- Buttons: Submit and Cancel.

**Use Case 2: Searching for Books Step-by-Step Process:**

1. **Start Point:**
   The user logs in to the platform and is taken to the Dashboard. The dashboard includes options like Browse Books, Search Books, View Account, and Settings.

2. **User Action:**
   The user clicks on the Search Books option, which directs them to the Search Page.

3. **System Response:**
   - The system displays a search bar with the label "Search for Books" and a button labeled Search.
   - There are also additional filters like Genre, Author, and Language available next to the search bar.

4. **User Action:**
   The user enters a search term (e.g., “Artificial Intelligence”) and clicks the Search button.

5. **System Response:**
   The system displays a list of matching books based on the search query and selected filters.

   Each book in the list shows:
   - **Title**
   - **Author**
   - **Genre**
   - **A brief description**
   - **Read Now** button for immediate access.

6. **User Action:**
   The user clicks on a book title from the search results to view more details.

7. **System Response:**
   The system opens the Book Detail Page where the user can:

   - See the full book description.
   - Read the book (clicking on Read Now button).
   - Optionally, leave a review or rating.

**Navigational Path:**

Dashboard → Search Books → Search Page → Search Results → Book Detail Page → Reading Page.

**Display Mock-up:**

Search Page:

- **Search Bar:** For entering search queries.
- **Filters:** Drop-down options for Genre, Author, and Language.
- **Search Button:** To trigger the search.
- **List of Books:** Displaying search results.
- **Book Title, Author, Genre, and Description:** Information for each book.
- **Read Now Button:** To start reading the selected book.

**b. User Effort Estimation**

**Case 1: User Registration and Login**

1. **Navigation (clicks)**:
    - From the Homepage, the user clicks on the **Register** button: **1 click**.
    - After filling out the form, they click **Submit** to complete the registration: **1 click**.
    - **Total navigation effort**: **2 clicks**.
2. **Data Entry (typing)**:
    - Full Name: **10-20 keystrokes**.
    - Username: **6-12 keystrokes**.
    - Email Address: **12-20 keystrokes**.
    - Password: **8-16 keystrokes**.
    - Confirm Password: **8-16 keystrokes**.
    - **Total typing effort**: **44-72 keystrokes**.

**Total effort for User Registration**:

- **2 clicks** and **44-72 keystrokes**.

### Case 2: Searching for Books

1. **Navigation (clicks)**:
    - From the Dashboard, the user clicks the **Search Books** option: **1 click**.
    - On the Search Page, they click the **Search** button: **1 click**.
    - When browsing the results, they click on a book title to view more details: **1 click**.
    - **Total navigation effort**: **3 clicks**.
2. **Data Entry (typing)**:
    - Search term (e.g., "Artificial Intelligence"): **20-30 keystrokes**.
    - Filters (if applicable): **5-10 keystrokes for each filter** (1 or 2 filters, so around **10-20 keystrokes**).
    - **Total typing effort**: **25-40 keystrokes**.

**Total effort for Searching for Books**:

- **3 clicks** and **25-40 keystrokes**.

==========

Report #1: Full Report

**5. System Architecture and System Design**

**a. Identifying Subsystems**

The library website is designed using a modular architecture, dividing its functionality into distinct subsystems that interact to provide a seamless user experience. These subsystems ensure efficiency, maintainability, and scalability. Below is a breakdown of the key subsystems:

**a.1. User Management Subsystem**

Purpose: Handles user authentication, registration, and role management.
Key Features:

- User sign-up and login
- Profile management (name, email, preferences, etc.)
- Role-based access (e.g., librarian vs. regular user)

Interactions with Other Subsystems:

- Book Catalog Subsystem (Users search for books)
- Checkout & Return Subsystem (Users check out books)
- Notification Subsystem (Users receive alerts on due dates)

### a.2. Book Catalog Subsystem

Purpose: Stores and organizes book data, allowing users to search and browse books.
Key Features:

- Book search by title, author, genre, or ISBN
- Availability status (checked out or available)
- Book details and descriptions

Interactions with Other Subsystems:

- Checkout & Return Subsystem (Changes book status upon checkout)
- Reservation & Hold Subsystem (Handles book reservations)

### a.3. Checkout & Return Subsystem

Purpose: Manages book borrowing and returning, enforcing due dates and tracking overdue items.
Key Features:

- Book checkout with due dates
- Book return processing
- Overdue fine calculations

Interactions with Other Subsystems:

- Notification Subsystem (Alerts users about due dates and fines)
- User Management Subsystem (Tracks borrowed books per user)

### a.4. Reservation & Hold Subsystem

Purpose: Allows users to place holds on books that are currently checked out.
Key Features:

- Users can place a hold on a book
- System notifies users when the book is available
- Automatic hold expiration if not picked up

Interactions with Other Subsystems:

- Book Catalog Subsystem (Updates book availability)
- Notification Subsystem (Sends hold availability alerts)

### a.5. Administrative Subsystem

Purpose: Enables librarians to manage books, users, and system operations.
Key Features:

- Add, update, or remove books from the catalog
- Manage user accounts
- View reports on checkouts, returns, and overdue books

Interactions with Other Subsystems:

- Book Catalog Subsystem (Modifies book records)
- User Management Subsystem (Handles user roles and permissions)

### a.6. Notification & Messaging Subsystem

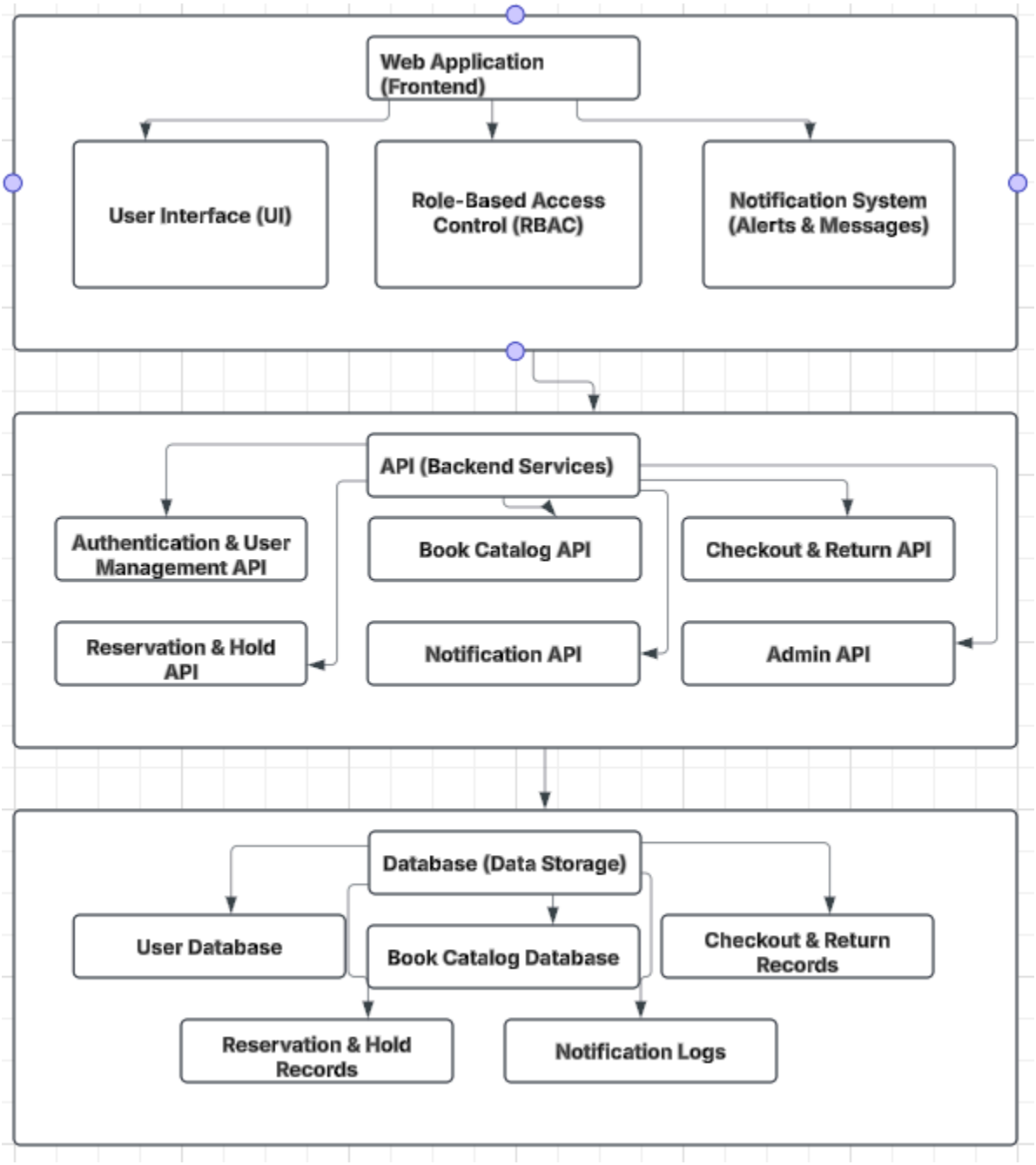
Purpose: Sends notifications to users regarding book availability, due dates, overdue books, and system updates.
Key Features:

- Email/SMS alerts for due dates, returns, and holds
- System announcements for library events and policy changes

Interactions with Other Subsystems:

- Checkout & Return Subsystem (Sends due date reminders)
- Reservation & Hold Subsystem (Alerts users about available holds)

**a.7 Diagram**




**b. Architecture Styles: Client-Server Architecture**

**Overview**:
The Digitized Library Management System follows a Client-Server Architecture to separate the front-end user interface from back-end data processing. The client handles

user interactions, while the server processes requests, manages data, and sends responses. This setup ensures modularity, scalability, and ease of maintenance.

**Components:**

**b.1. Client (Front-End)**:
The client is a web application accessible via desktops and mobile devices, offering a simple, user-friendly interface.

**Key responsibilities**:

- **User Authentication**: Manage registration, login, and account security.
- **Book Search**: Users can search, view, and filter books by title, author, ISBN, and more.
- **Checkout & Reservations**: Allows checking out books, reserving unavailable ones, and tracking borrowing history.
- **Notifications**: Sends reminders for due dates and availability updates.
- **Account Management**: Users can manage their borrowing history, wishlist, and settings.

**b.2. Server (Back-End)**:

The server processes requests, interacts with the database, and sends necessary data back to the client.

**Key responsibilities**:

- **Request Processing**: Handles book searches, user logins, and check-out requests.
- **Database Interaction**: Updates and retrieves data like book records and user info.
- **Real-Time Availability**: Ensures up-to-date availability and overdue statuses.
- **Security**: Manages authentication, input validation, and data protection.
- **Notifications**: Sends due date and overdue alerts.

**Communication**:
The client and server communicate via HTTP/HTTPS using RESTful APIs to request and deliver data efficiently.

**Benefits:**

- **Centralized Management**: Server handles data and processing, maintaining consistency across tasks.

- **Scalability**: Easy to scale by adding servers, and client updates can be deployed independently.
- **Security**: Sensitive data is securely stored on the server, with encryption and protection measures.
- **Separation of Concerns**: Client focuses on user interaction, while the server handles data management and processing.

**Real-World Example**:
An online retail website shows this architecture, where the client (browser) allows users to browse products, while the server handles user authentication, payment, and inventory management.

**Conclusion**:
The Client-Server Architecture ensures that the Digitized Library Management System is scalable, secure, and responsive, providing a seamless experience for both users and librarians.

**c. Mapping Subsystems to Hardware**

The Library Management System runs on multiple computers. The system is divided into three main parts:

- **Server Subsystem:** This runs on a dedicated server that handles all the main tasks—managing the database, processing book loans, handling user accounts, and sending out notifications. It also hosts the web application and manages communication between the database and the client devices.
- **Database Subsystem:** The database runs on a separate server to improve performance and security. It stores all the important information, like user accounts, book details, borrowing history, and reservation records.
- **Client Subsystem:** Users can access the system through web browsers on various devices, including desktops, smartphones, and tablets. Library staff use desktop computers to manage books, process loans, and handle user accounts.

**d. Connectors and Network Protocols**

- **HTTPS:** The main protocol for accessing the site, this is a commonly used protocol for anything website-related. This is compatibly with various devices and ensures proper data encryption.

- **Database Connector:** This will allow communication between the user and the database in order to retrieve information on books they would like to borrow and update the status of it.

- **Authentication Connector:** This will be used for verifying credentials when they log in. This also allows us to determine whether the person who logged in has the permissions of a regular user or a librarian who has access to additional features.

**e. Global Control Flow**

- **Execution orderliness:** The system is event-driven, meaning it doesn't follow a strict step-by-step process. Users can interact with the system in any order—searching for books, making reservations, or returning borrowed items whenever they need to. Library staff can update book records or manage user accounts at the same time without interrupting users' actions.
- **Time dependency:** The system works on an event-response basis, so it doesn't rely on real-time processes. However, it does use time-based triggers to send reminders for due dates, overdue notifications, and reservation updates. There aren't any strict time constraints for other functions.

**f. Hardware Requirements**

- **Screen Display:** The website can be accessed on multiple devices such as PCs and mobile devices. Thus it will require hardware that is capable of displaying text and images to a screen. The exact display requirement will vary depending on the device but any system capable of running html should have the capability to display the software.

- **Communication Network:** A stable internet connection is needed to access the website online. The users only need a minimum of 1 Mbps for a stable connection.

- **Disk Storage:** The application requires a separate server to host the application files as well as the database for user accounts, book data, check in/out logs, and backup data. The amount can range anywhere between 10-50 gb depending on factors such as storing book images and the amount of backups.

- **Database Server:** The application will require a database server to process requests and compute database operations.

- **Security Measures:** Adequate security measures such as firewalls and encryption will safeguard any user data that is logged in the system.

==========

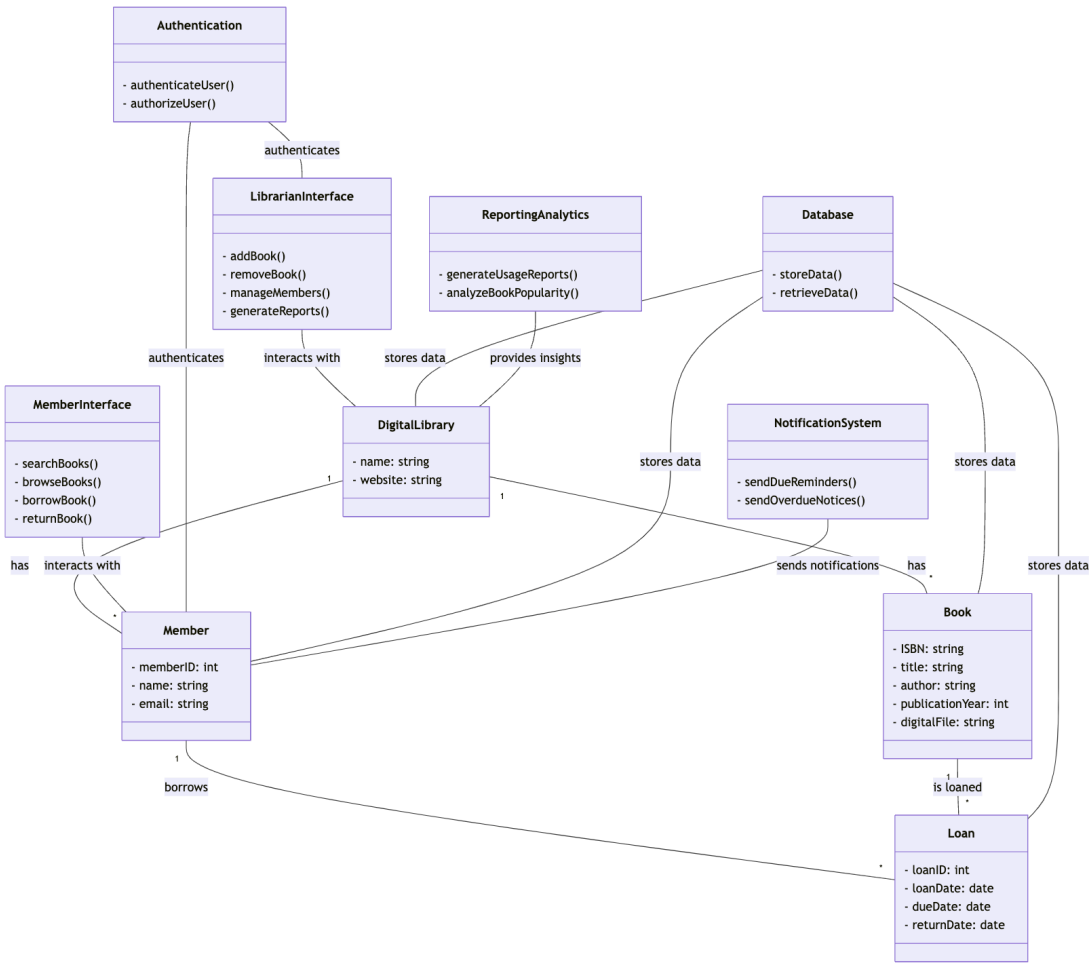
Report #2 - Part 1

## 6. Domain Analysis

### a. Conceptual model

In this section, we present the Conceptual Model for the Digitized Library Management System. It defines the key components of the system, such as DigitalLibrary, Member, Book, and Loan, and shows how they are related. This model outlines important details like Member IDs, Book ISBNs, and Loan due dates. It also highlights system operations like authentication, database management, notifications, and reporting, ensuring everything works smoothly for efficient library management.

## Domain Model (update)



### I. Concepts Definition

| Responsibility Description | Type (D=Doing, K=Knowing) | Concept Name |
|---|---|---|
| Stores and manages member information (ID, name, contact details) | ➔ K | Member |
| Stores and manages book information (ISBN, title, author, publication year, digital file) | ➔ K | Book |
| Tracks the availability status of digital book copies | ➔ K | Book Copy |
| Manages the loan and return of digital books | ➔ D | Loan Management |
| Provides an interface for members to search, browse, and borrow digital books | ➔ D | Member Interface |
| Provides an interface for librarians to manage books, members, and loans | ➔ D | Librarian Interface |
| Handles user authentication and authorization | ➔ D | Authentication/Authorization |
| Stores and retrieves data related to members, books, and loans | ➔ D | Database Management |
| Sends notifications to members regarding due dates, overdue books, and other relevant information | ➔ D | Notification System |
| Provides reporting and analytics on library usage and book popularity | ➔ D | Reporting and Analytics |

## II. Association Definitions

| Concept | Attribute | Description |
|---|---|---|
| **Book Catalog** | ➔ BookID<br>➔ Title, Author<br>➔ ISBN<br>➔ Genre<br>➔ PublishedDate<br>➔ CopiesAvailable | Stores and organizes book information for browsing and searching. |
| **User Account** | ➔ UserID, Name<br>➔ Email<br>➔ PhoneNumber<br>➔ Address<br>➔ MembershipType<br>➔ JoinDate | Manages user details, allowing users to borrow books and track their history. |
| **Book Borrowing** | ➔ BorrowID<br>➔ UserID<br>➔ BookID<br>➔ BorrowDate<br>➔ DueDate<br>➔ ReturnDate<br>➔ Status | Tracks which books are borrowed, due dates, and return status. |
| **Reservations** | ➔ ReservationID<br>➔ UserID, BookID<br>➔ RequestDate<br>➔ ExpirationDate<br>➔ Status | Allows users to reserve books that are currently checked out. |
| **Fines & Payments** | ➔ FineID<br>➔ UserID<br>➔ Amount<br>➔ DueDate<br>➔ Status<br>➔ PaymentDate | Tracks overdue book fines and payments made by users. |
| **Librarian Management** | • LibrarianID<br>• Name<br>• Email<br>• PhoneNumber | Manages librarian details and their roles in the system. |

| | | |
|---|---|---|
| | • Role<br>• HireDate | |
| **Notifications** | • NotificationID<br>• UserID<br>• Type<br>• Message<br>• CreatedAt<br>• ReadStatus | Sends alerts for due dates, reservations, and library updates. |
| **Reviews & Ratings** | • ReviewID<br>• UserID<br>• BookID<br>• Rating<br>• Comment<br>• CreatedAt | Allows users to leave reviews and ratings for books. |
| **Library Events** | • EventID<br>• Title<br>• Description<br>• Date<br>• Time<br>• Location | Displays upcoming library events such as book readings and workshops. |
| **Digital Resources** | • ResourceID<br>• Title<br>• Type<br>• URL<br>• AccessLevel<br>• ExpirationDate | Provides access to e-books, research papers, and online databases. |

### III. Attribute Definitions

| Association | Entities Involved | Multiplicity | Description |
|---|---|---|---|
| **User Borrows Book** | *User* → *Book* | 1..* → 0..* | A user can borrow multiple books over time, and a book can be borrowed by multiple users at different times (but |

| | | | |
|---|---|---|---|
| | | | only one user at a time). |
| **User Reserves Book** | *User → Reservation* | 1 → 0..* | A user can place multiple reservations, but each reservation belongs to only one user. |
| **Reservation Holds Book** | *Reservation → Book* | 1 → 1 | Each reservation corresponds to a single book, and each book can have at most one active reservation at a time. |
| **Librarian Manages Book Inventory** | *Librarian → Book* | 1 → 0..* | A librarian can manage multiple books in the library catalog, but each book must be managed by at least one librarian. |
| **User Has an Account** | *User → Account* | 1 → 1 | Each user has a single account, and each account belongs to one user. |
| **Book Belongs to Category** | *Book → Category* | 0..* → 1 | A book belongs to exactly one category, but a category can contain multiple books. |
| **User Pays Fine** | *User → Fine* | 1 → 0..* | A user can have multiple fines, but each fine belongs to only one user. |
| **Fine is Associated with Book** | *Fine → Book* | 1 → 1 | A fine is always associated with a specific book. |

| | | | |
|---|---|---|---|
| **Librarian Manages User Accounts** | *Librarian → User* | 1 → 0..* | A librarian can manage multiple user accounts, but each user account is managed by at least one librarian. |

### IV. Traceability Matrix

| **Use Case** | **Domain Concepts Involved** |
|---|---|
| UC-1: Search Books | *User, Book, Category* |
| UC-2: Check Book Availability | *Book, Reservation, User* |
| UC-3: Manage Book Checkout | *User, Book, Librarian, Account, Fine* |
| UC-4: Access Help Section | *User, Librarian, HelpRequest* |
| UC-5: Manage User Account | *User, Account, BorrowingHistory* |
| UC-6: Reserve Book | *User, Account, BorrowingHistory* |
| UC-7: Manage Library Inventory | *Librarian, Book, Inventory* |
| UC-8: Pay Fine | *User, Fine, Book, Librarian* |

**b. System Operation Contracts**

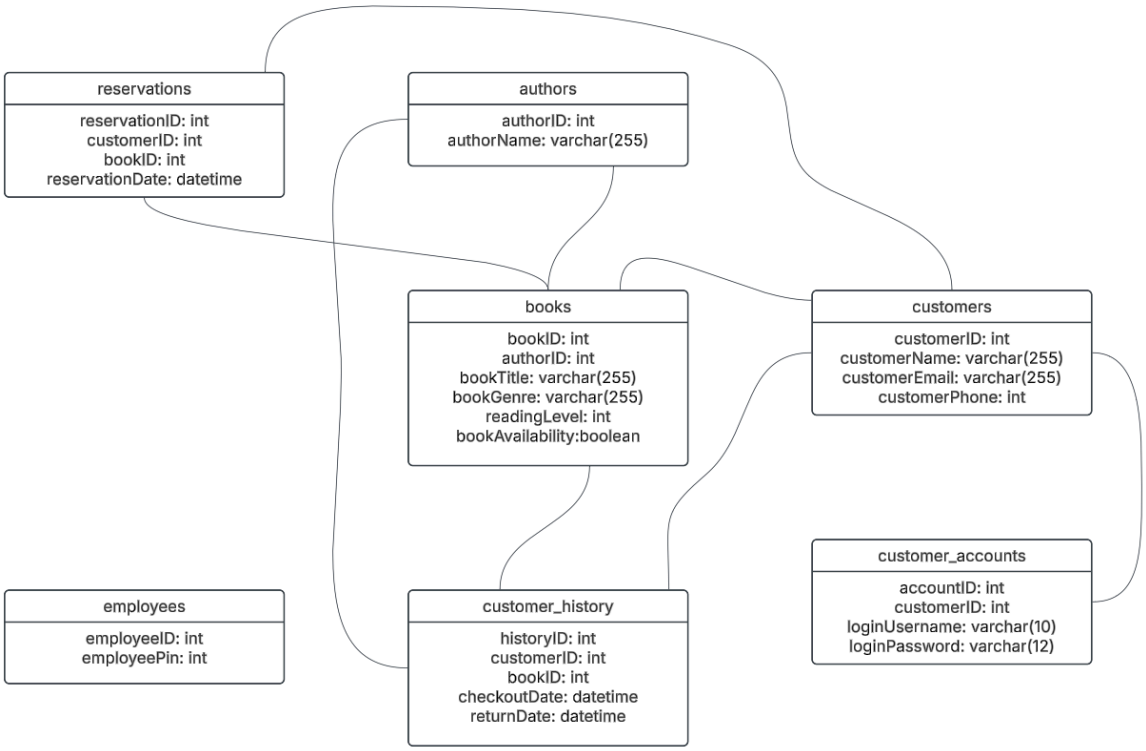
| **Operation** | **Search Books** |
|---|---|
| Use Case | UC1 |
| Preconditions | User fills in the field for name, title and/or ISBN.<br>User applies optional filters<br>User clicks the search button |
| Postconditions | A curated list of books matching the search specification is listed in order of best to worst match. |

| **Operation** | **Manage Book Checkout** |
|---|---|
| Use Case | UC3 |
| Preconditions | User is logged in<br>User has navigated to a specific book<br>User checks in/out the book |
| Postconditions | The book's state is toggled between borrowed/available based on the user's borrowing relation to the book |

| **Operation** | **Reserve Book** |
|---|---|
| Use Case | UC6 |
| Preconditions | User is logged in<br>User has navigated to a specific book<br>User clicks the reserve button |
| Postconditions | The book is reserved and becomes unselectable for other users. |

| **Operation** | **Manage Library Inventory** |
|---|---|
| Use Case | UC7 |
| Preconditions | User is logged in<br>User has the librarian role<br>The user opens the librarian interface |
| Postconditions | The user is taken to the librarian interface where they can add/remove books |

### c. Data Model and Persistent Data Storage



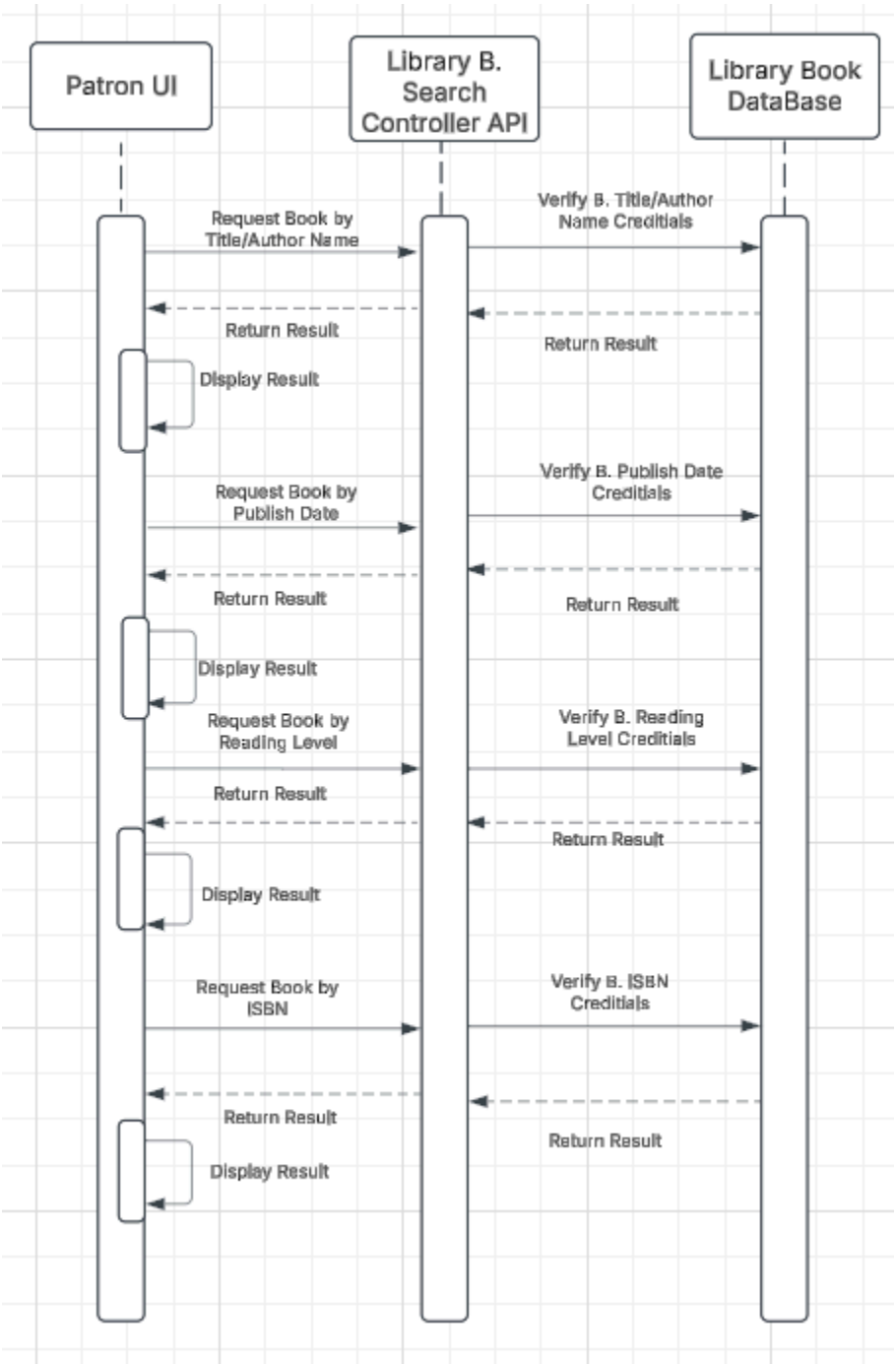

Our system will require persistent data storage to manage a library system effectively. The data will be stored in designated tables based on the entity type. For **books**, the system will store essential information such as a unique book ID, author ID, title, genre, reading level, and availability status. **Authors** will have their own table, containing a unique author ID and name. For **customers**, the system will retain details such as customer ID, name, email, and phone number, along with their login credentials stored in a separate table for security, including a unique account ID, customer ID, username, and password. The system will also track **customer history**, recording each book checked out and returned, with details such as history ID, customer ID, book ID, checkout date, and return date. Additionally, **reservations** will be stored, including reservation ID, customer ID, book ID, and reservation date. For **employees**, the system will store a unique employee ID and a PIN for authentication. All this data will reside in a relational database, such as MySQL, hosted on our web server, ensuring structured data storage, data integrity through foreign keys, and support for complex queries to manage library operations efficiently.

==========

Report #2 - Part 2

## 7. Interaction Diagrams

### a. UC-1: SearchBooks Diagram



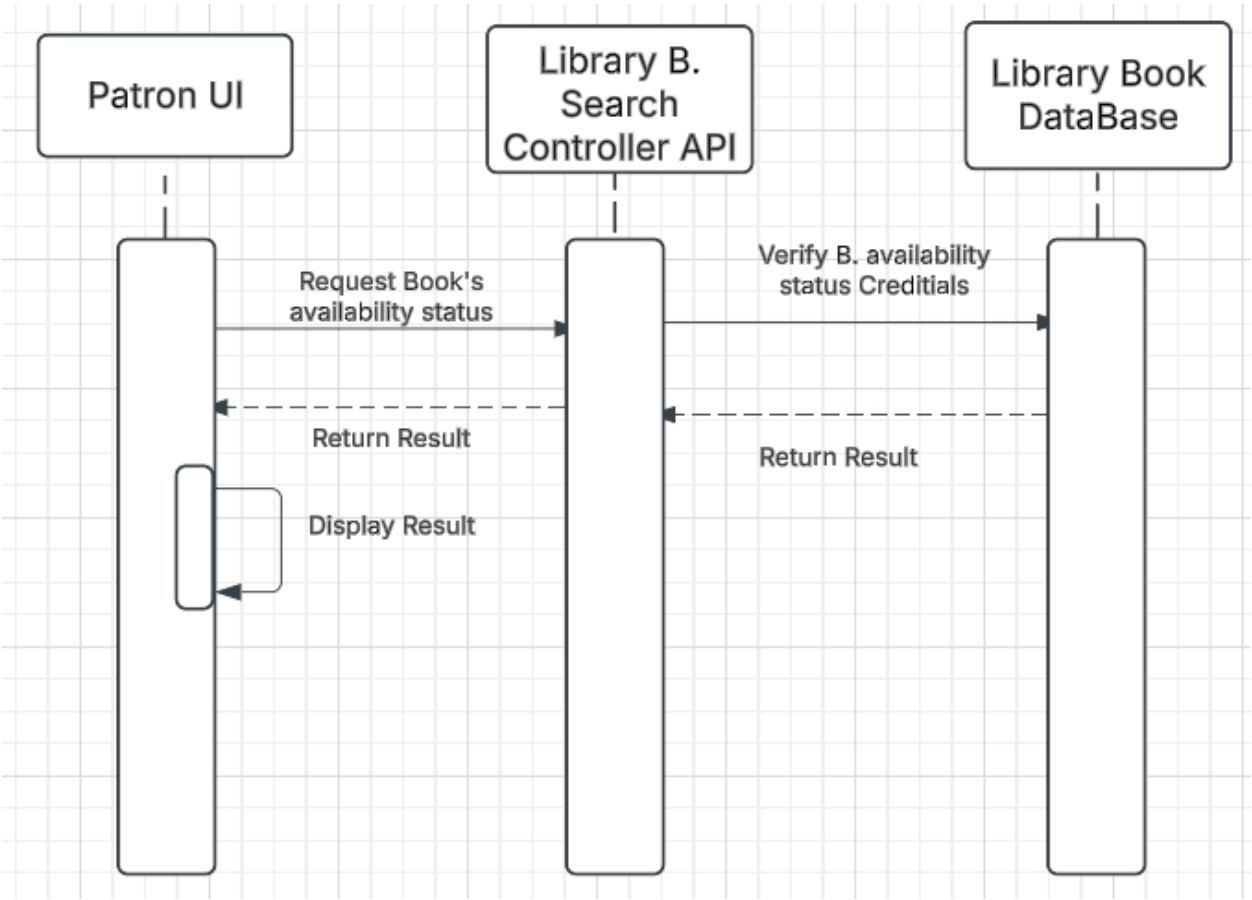

The diagram demonstrates the interactions in UC-1: SearchBooks. Below is a description of the interactions laid out in the diagram:

- The user accesses the system and initiates a book search by selecting a search criterion (e.g., title, author name, ISBN, etc.).
- The system interacts with the "Library B. Search Controller API" to process the search request.
- The API queries the "Library Book Database" (SQL-based) using the provided search criteria.
- The database returns the search results to the API, which then filters the results based on additional criteria (e.g., reading level, publishing date) if specified by the user.
- The filtered results are displayed to the user.

The design principles employed in this sequence are the Expert Doer Principle and Low Coupling Principle.

- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Book Database" component, which is responsible for storing and retrieving book data. It possesses the necessary information (e.g., book titles, authors, ISBNs, reading levels, publishing dates) to execute queries efficiently.
    - The principle is also employed in the "Search Controller API," which is responsible for processing search requests and applying filters. It has the knowledge of how to interact with the database and how to filter results based on user input.
- The Low Coupling Principle states that components should have minimal dependencies on each other to promote flexibility and ease of modification.
    - The principle is employed by ensuring that the "Search Controller API" and the "Library Book Database" are loosely coupled. The API interacts with the database through well-defined interfaces (e.g., SQL queries), allowing the database to change its internal structure without affecting the API.

### b. UC-2: CheckBookAvailability Diagram




The diagram demonstrates the interactions in UC-2: CheckBookAvailability. Below is a description of the interactions laid out in the diagram:

- The user initiates a request to check the availability of a specific book, either directly or as an extension of UC-1: SearchBooks.
- The system interacts with the "Library B. Search Controller API" to process the availability request.
- The API queries the "Library Book Database" (SQL-based) to retrieve the real-time availability status of the book (e.g., whether it is currently checked out, on hold, or available).
- The availability status is returned to the API, which then formats and displays the information to the user.

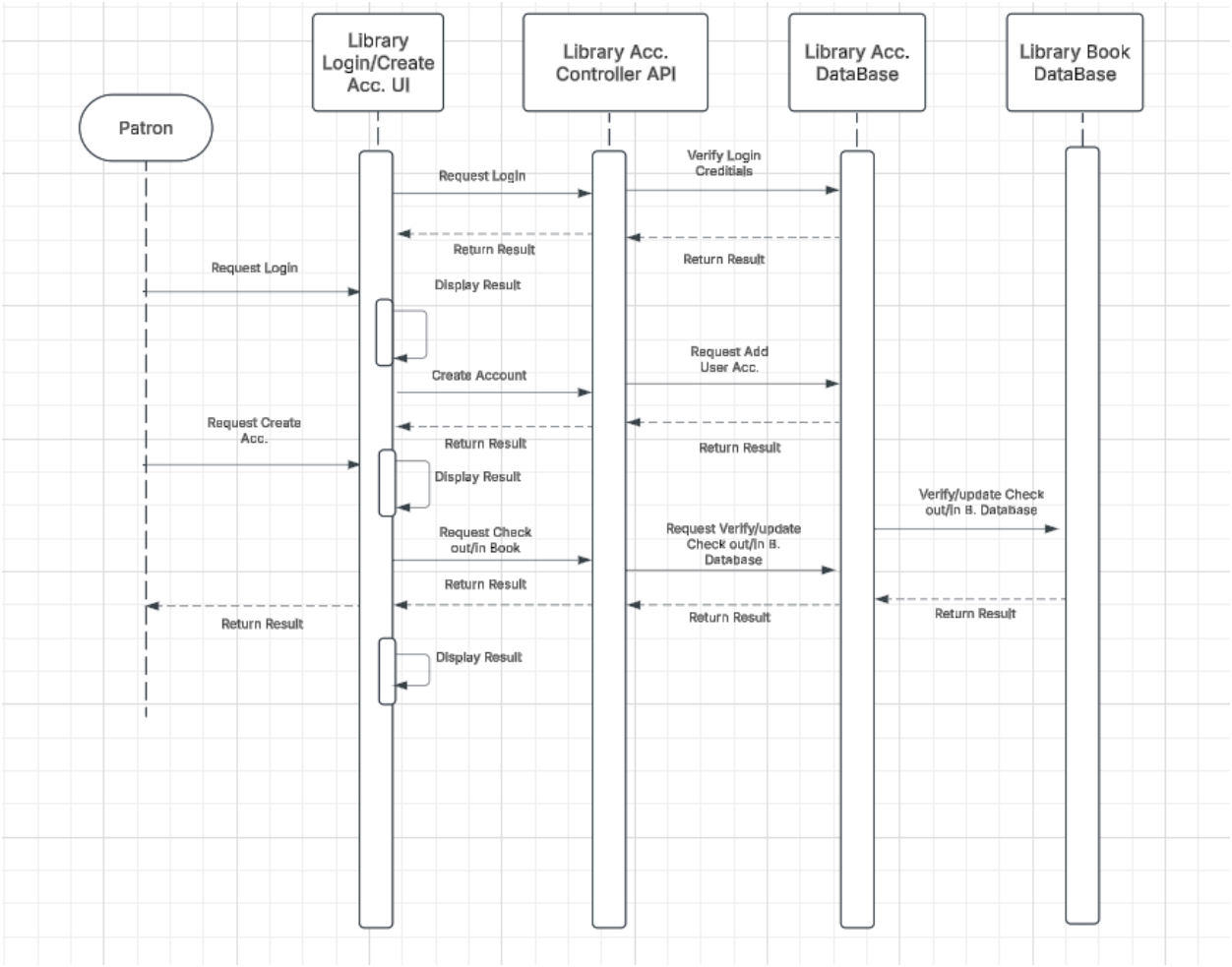
The design principles employed in this sequence are the Expert Doer Principle.

- Expert Doer Principle suggests that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.

- The principle is employed in the "Library Book Database" component, which is responsible for storing and retrieving a book's availability. It possesses the necessary information (e.g., book IDs, checkout status, hold status) to determine the real-time availability of a book.
- The principle is also employed in the "Search Controller API," which is responsible for processing availability requests. It has the knowledge of how to interact with the database and how to format the availability status for display to the user.

By employing these principles, the system ensures that responsibilities are clearly defined and related functionality is grouped logically, making the system easier to maintain and further develop. The "Library Book Database" acts as our expert in this case.

### c. UC-3: ManageBookCheckout Diagram




The diagram demonstrates the interactions in UC-3: ManageBookCheckout. Below is a description of the interactions laid out in the diagram:

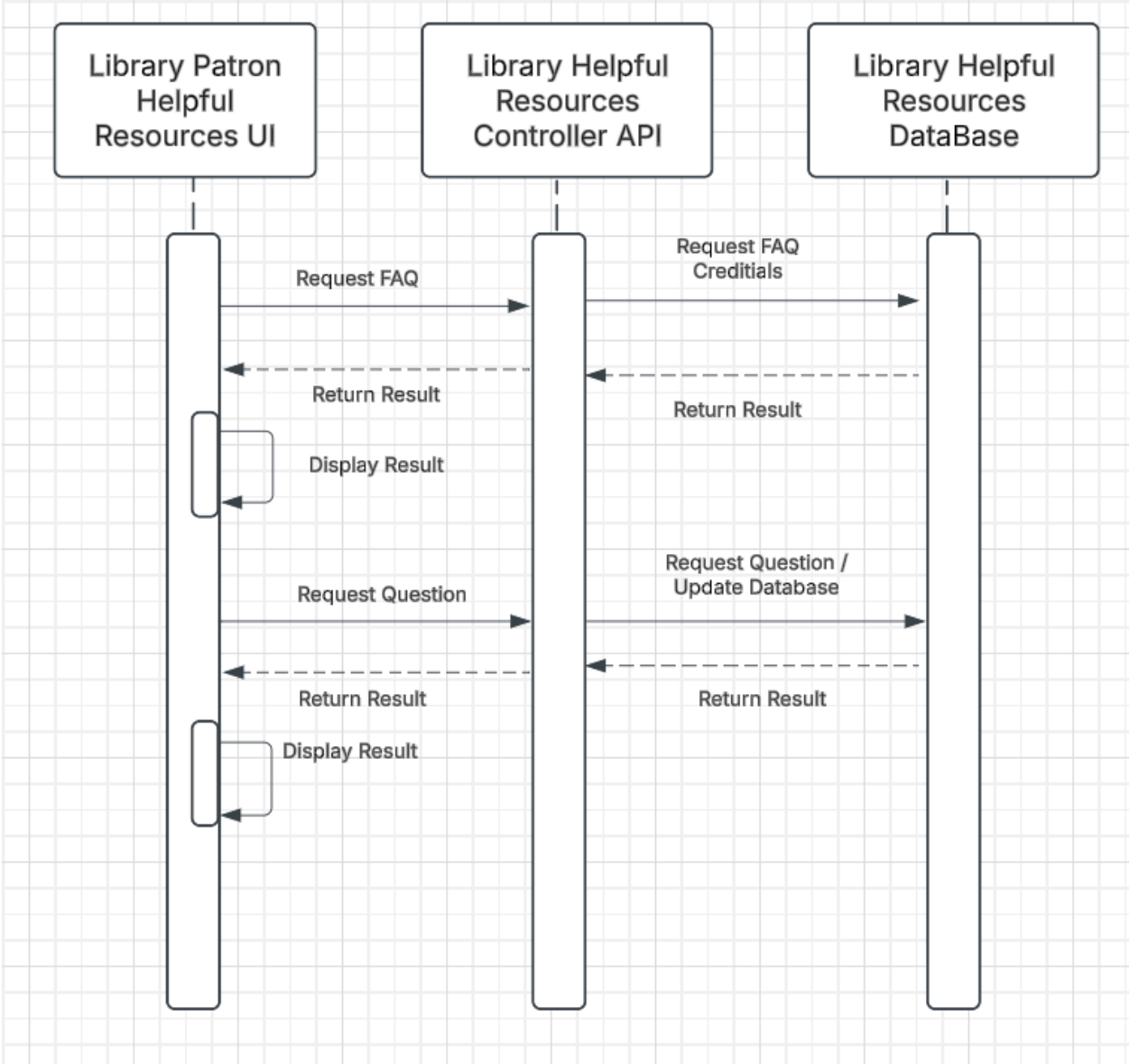
- The system interacts with the "Library Login/Create Account UI" to verify the user's identity. If the user does not have an account, they are prompted to create one.
- Once authenticated (or after account creation), the system uses the "Library B. Search Controller API" to process the checkout or check-in request.
- The API interacts with the "Library Book Database" (SQL-based) to update the book's status (e.g., marking it as checked out or checked in).
- The API also interacts with the "Library Account Database" (SQL-based) to update the user's account with the relevant book.
- The system confirms the "Check Out" to the user.

The design principles employed in this sequence are the Expert Doer Principle and the Low Coupling Principle.

- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Book Database" component, which is responsible for storing and updating book statuses (e.g., checked out, checked in). It possesses the necessary information to manage book availability and update records accordingly.
    - The principle is also employed in the "Library Account Database," which is responsible for storing and updating user account information (e.g., books checked out by the user, user credentials). It has the knowledge to manage user-specific book transactions and account creation.
    - The principle is further applied in the "Library Login/Create Account UI," which is responsible for handling user authentication and account creation. It has the knowledge to verify user credentials or guide users through the account creation process.
    - Finally, the principle is applied in the "Search Controller API," which is responsible for coordinating the checkout/check-in process. It has the knowledge of how to interact with both the book and account databases to ensure the transaction is completed successfully.
- Low Coupling Principle states that components should have minimal dependencies on each other to promote flexibility and ease of modification.
    - The principle is employed by ensuring that the "Search Controller API" acts as an intermediary between the user interface, the book database, and the account database. This reduces direct dependencies between the databases and the UI, allowing each component to operate independently.
    - The principle is also applied by separating the authentication and account creation process (handled by the "Library Login/Create Account UI") from the checkout/check-in logic (handled by the "Search Controller API"). This ensures that changes to the authentication or account creation process do not affect the checkout/check-in functionality.
    - Additionally, the "Library Book Database" and "Library Account Database" are loosely coupled, meaning changes to one database (e.g., adding new fields to the account database) do not directly impact the other.

By employing these principles, the system ensures that responsibilities are clearly defined and components remain loosely coupled, making the system easier to maintain and further develop. The "Library Book Database" and "Library Account Database" act as experts, while the "Search Controller API" and "Library Login/Create Account UI" ensure low coupling by coordinating interactions between components.

### d. UC-4: AccessHelpSection diagram




The diagram demonstrates the interactions in UC-4: AccessHelpSection. Below is a description of the interactions laid out in the diagram:

- The user accesses the system and navigates to the help section to seek assistance.
- The system interacts with the "Library Patron Helpful Resources UI" to display available assistance resources.
- The UI communicates with the "Library Patron Helpful Resources API" to retrieve relevant data.
- The API queries the "Library Patron Helpful Resources Database" (SQL-based) to fetch a list of frequently asked questions (FAQ) or to process a user's question sent to a librarian.

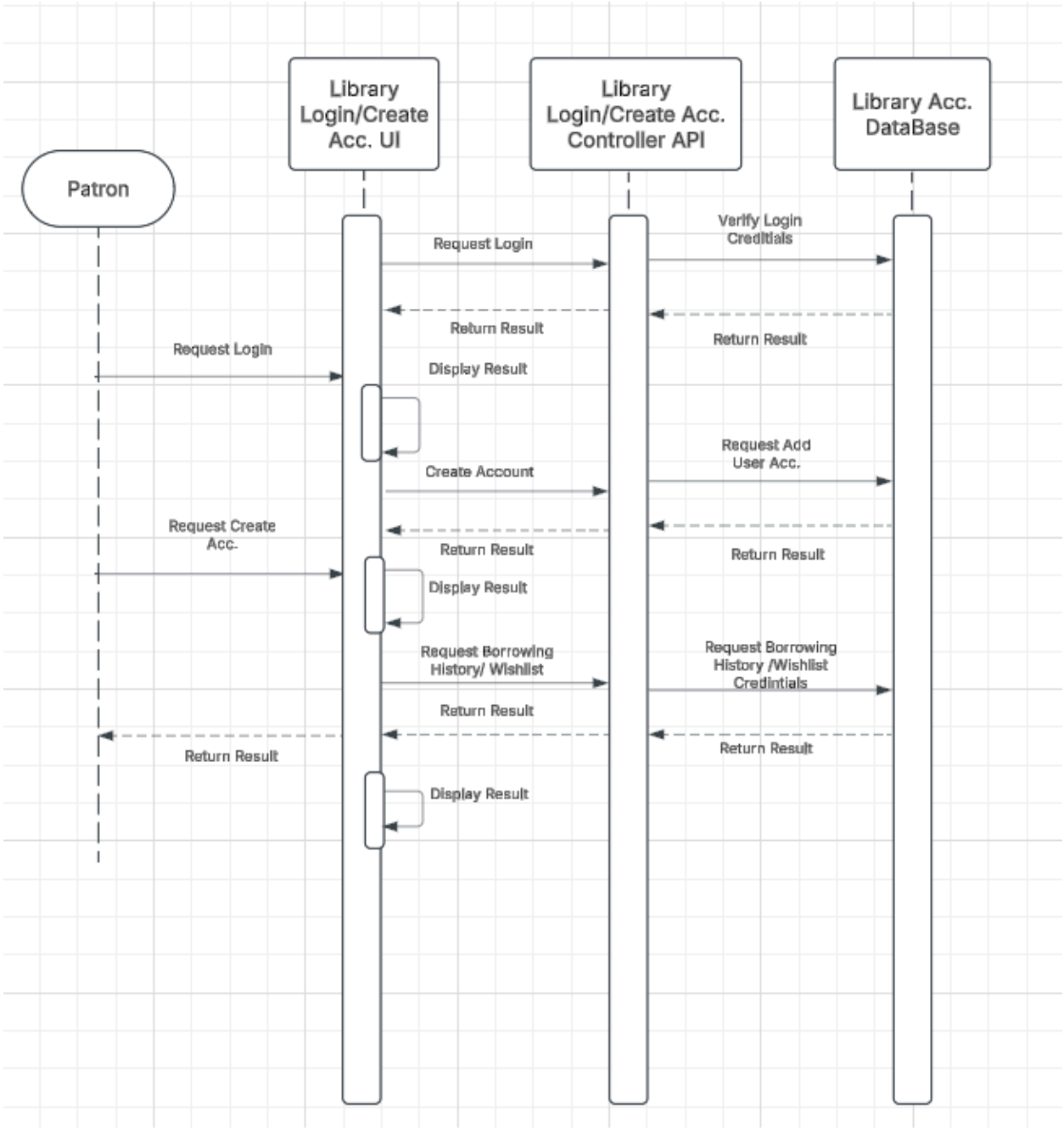
- The database returns the requested data (e.g., FAQ list or confirmation of the submitted question) to the API.
- The API formats the data and sends it back to the UI, which then displays it to the user.

The design principles employed in this sequence are the Expert Doer Principle and the Low Coupling Principle.

- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Patron Helpful Resources Database" component, which is responsible for storing and retrieving help-related data (e.g., FAQs, submitted questions). It possesses the necessary information to execute queries efficiently and manage help resources.
- Low Coupling Principle states that components should have minimal dependencies on each other to promote flexibility and ease of modification.
    - The principle is applied by separating the UI layer ("Library Patron Helpful Resources UI") from the data processing layer ("Library Patron Helpful Resources API"). This ensures that changes to the UI (e.g., redesigning the help section layout) do not impact the API or database logic.

By employing these principles, the system ensures that responsibilities are clearly defined and components remain loosely coupled, making the system easier to maintain and further develop. The "Library Patron Helpful Resources Database" acts as the expert for help-related data, while the "Library Patron Helpful Resources API" ensures low coupling by coordinating interactions between the UI and the database.

### e. UC-5: ManageUserAccount Diagram




The diagram demonstrates the interactions in UC-5: ManageUserAccount. Below is a description of the interactions laid out in the diagram:

- The system interacts with the "Library Login/Create Account UI" to verify the user's identity. If the user does not have an account, they are prompted to create one.
- The system interacts with the "Library Login/Create Account UI" to handle account creation or authentication.
- Once authenticated, the system uses the "Library B. Search Controller API" to process user requests related to borrowing history or wishlist management.

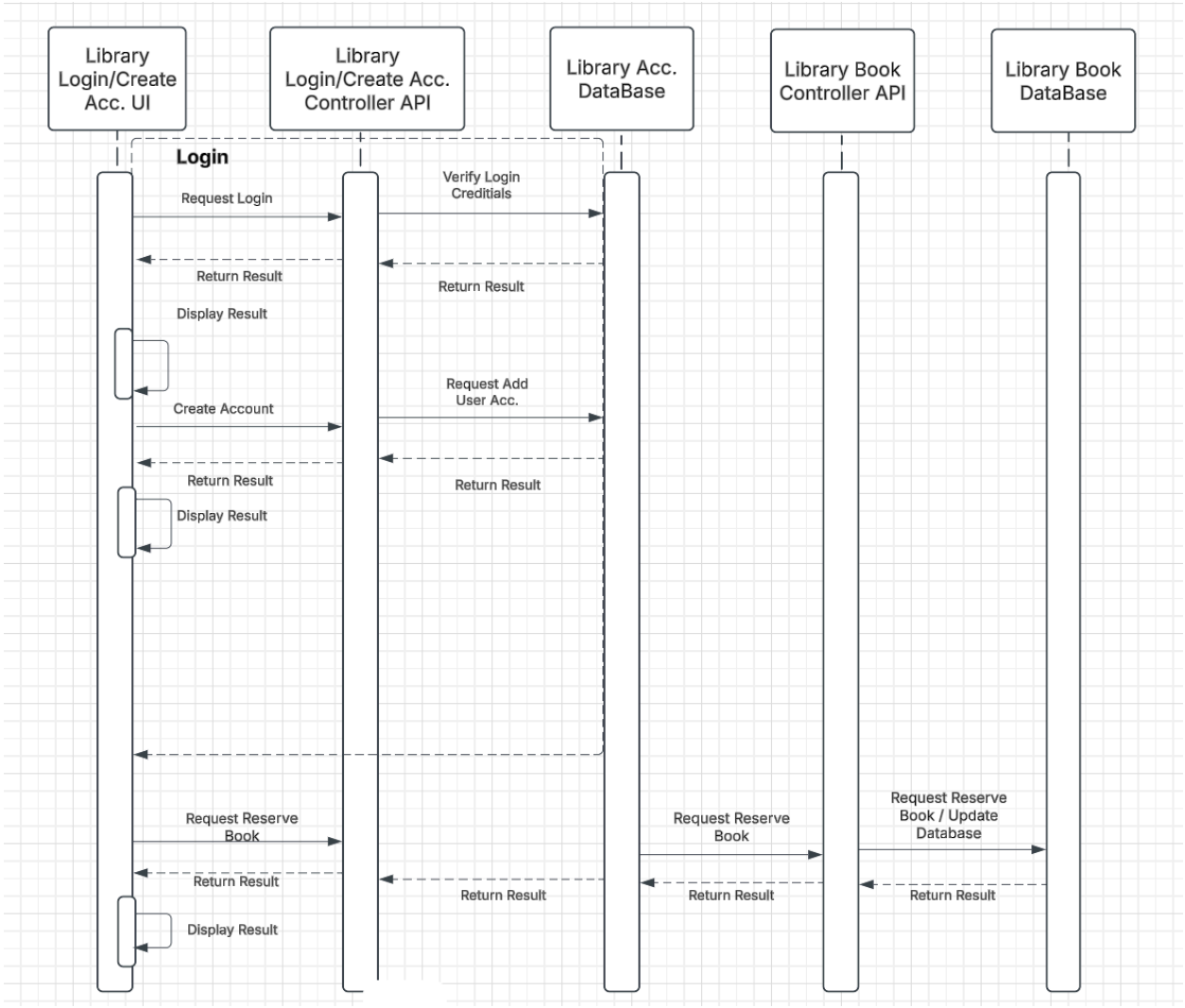
- The API interacts with the "Library Account Database" (SQL-based) to retrieve or update user-specific data (e.g., borrowing history, wishlist).
- For wishlist management, the API may also interact with the "Library Book Database" (SQL-based) to fetch book details (e.g., titles, authors, availability).
- The system displays the requested information (e.g., account details, borrowing history, wishlist) to the user.

The design principles employed in this sequence are the Expert Doer Principle and the High Cohesion Principle.

- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Account Database" component, which is responsible for storing and retrieving user account data (e.g., account credentials, borrowing history, wishlist). It possesses the necessary information to manage user-specific data efficiently.
    - The principle is also employed in the "Library Book Database," which is responsible for storing and retrieving book details. It has the knowledge to provide book-related information (e.g., titles, authors) for wishlist management.
- High Cohesion Principle states that related functionality should be grouped together within a module, class, or component to improve clarity and maintainability.
    - The principle is employed in the "Library Login/Create Account UI," which groups all functionality related to account creation and authentication. This ensures that tasks related to user accounts are centralized and logically organized.
    - The principle is further applied in the "Library Account Database," which groups all data storage and retrieval functionality related to user accounts, ensuring that database-related tasks are logically organized.

By employing these principles, the system ensures that responsibilities are clearly defined and related functionality is grouped logically, making the system easier to maintain and further develop. The "Library Account Database" and "Library Book Database" act as experts, while the "Library B. Search Controller API" and "Library Login/Create Account UI" ensure high cohesion by centralizing related tasks.

### f. UC-6: ReserveBook Diagram




The diagram demonstrates the interactions in UC-6: ReserveBook. Below is a description of the interactions laid out in the diagram:

- The system interacts with the "Library Login/Create Account UI" to verify the user's identity. If the user does not have an account, they are prompted to create one.
- Once authenticated, the system uses the "Library B. Search Controller API" to process the reservation request.
- The API interacts with the "Library Book Database" (SQL-based) to update the book's status (e.g., marking it as reserved).
- The API interacts with the "Library Book Database" (SQL-based) to check the book's availability status and place a reservation if the book is unavailable.
- The API also interacts with the "Library Account Database" (SQL-based) to update the user's account with the reservation details (e.g., book title, reservation date, expected availability date).

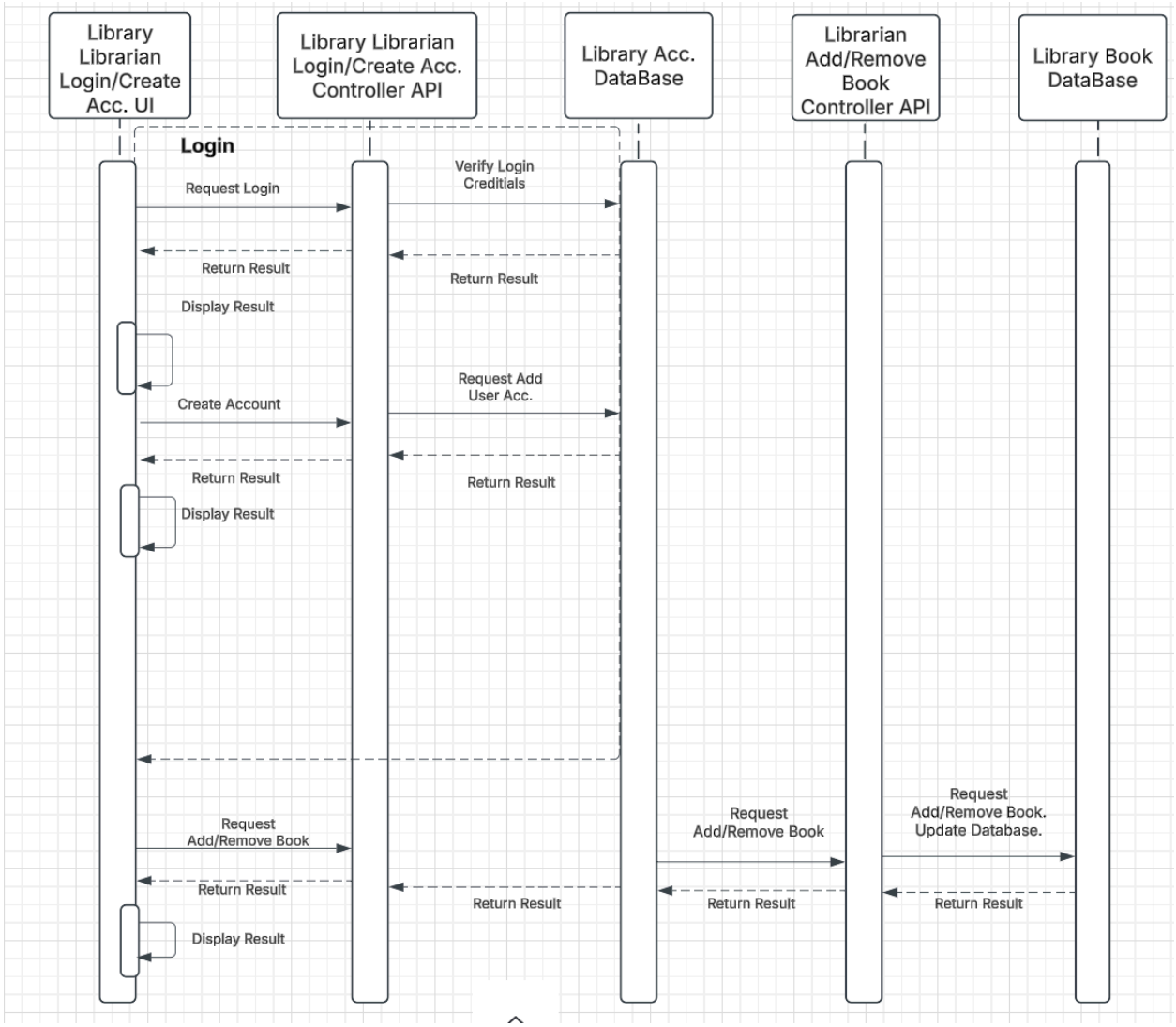
- The system confirms the reservation to the user and notifies them when the book becomes available.

The design principles employed in this sequence are the Expert Doer Principle and the Low Coupling Principle.

- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Book Database" component, which is responsible for storing and updating book statuses (e.g., checked out, checked in). It possesses the necessary information to manage book availability and update records accordingly.
    - The principle is also employed in the "Library Account Database," which is responsible for storing and updating user account information (e.g., reserved books).
    - The principle is demonstrated in the "Library Login/Create Account UI," which is responsible for handling user authentication and account creation. It has the knowledge to verify user credentials or guide users through the account creation process.
    - The principle is also applied in the "Library B. Search Controller API," which is responsible for coordinating the reservation process. It has the knowledge of how to interact with both the book and account databases to ensure the reservation is completed successfully.
- Low Coupling Principle states that components should have minimal dependencies on each other to promote flexibility and ease of modification.
    - The principle is employed by ensuring that the "Library B. Search Controller API" acts as an intermediary between the user interface, the book database, and the account database. This reduces direct dependencies between the databases and the UI, allowing each component to operate independently.
    - The principle is also applied by separating the authentication process (handled by the "Library Login/Create Account UI") from the reservation logic (handled by the "Library B. Search Controller API"). This ensures that changes to the authentication process do not affect the reservation functionality.
    - Additionally, the "Library Book Database" and "Library Account Database" are loosely coupled, meaning changes to one database (e.g., adding new fields to the account database) do not directly impact the other.

By employing these principles, the system ensures that responsibilities are clearly defined and components remain loosely coupled, making the system easier to maintain and further develop. The "Library Book Database" and "Library Account Database" act as experts for their respective data, while the "Library B. Search Controller API" ensures low coupling by coordinating interactions between components.

### g. UC-7: ManageLibraryInventory Diagram




The diagram demonstrates the interactions in UC-7: ManageLibraryInventory. Below is a description of the interactions laid out in the diagram:

- The librarian accesses the system and logs in using the "Library Librarian Login/Create Account UI." If the librarian does not have an account, they are prompted to create one.
- Once authenticated, the librarian can choose to add or remove books from the library system.
- The system uses the "Library Librarian Login/Create Account API" to process the librarian's requests.
- For adding and/or removing books, the API interacts with the "Library Book Database" (SQL-based) to insert new book records (e.g., title, author, ISBN, availability status).
- The system confirms the action (e.g., book added or removed) to the librarian.

The design principles employed in this sequence are the Expert Doer Principle and the High Cohesion Principle.

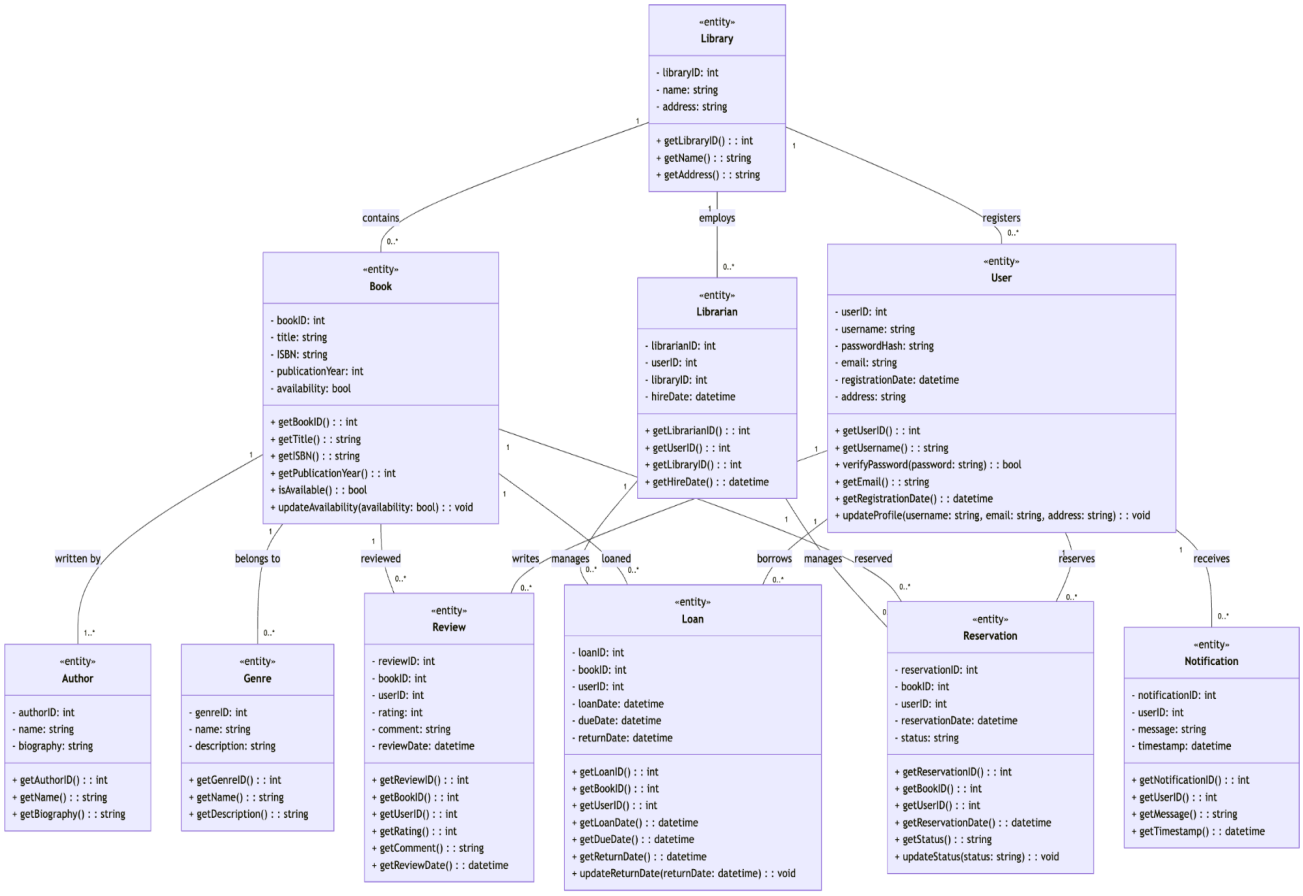
- Expert Doer Principle states that responsibility should be assigned to the class or component that has the most information or knowledge to fulfill that responsibility effectively.
    - The principle is employed in the "Library Book Database" component, which is responsible for storing and managing book records. It possesses the necessary information to add, remove, or update book data.
    - The principle is further applied in the "Library Librarian Login/Create Account API," which is responsible for coordinating the librarian's requests (e.g., adding or removing books). It has the knowledge of how to interact with both the book database and the librarian account database to fulfill these requests.
- High Cohesion Principle suggests that related functionality should be grouped together within a module, class, or component to improve clarity and maintainability.
    - The principle is employed in the "Library Librarian Login/Create Account UI," which groups all functionality related to librarian authentication and account creation. This ensures that tasks related to librarian accounts are centralized and logically organized.
    - The principle is also applied in the "Library Librarian Login/Create Account API," which groups all functionality related to managing library inventory (e.g., adding or removing books). This ensures that tasks related to inventory management are cohesive and well-structured.

By employing these principles, the system ensures that responsibilities are clearly defined and related functionality is grouped logically, making the system easier to maintain and further develop. The "Library Book Database" acts as an expert for their respective data, while the "Library Librarian Login/Create Account API" ensures high cohesion by centralizing related tasks.

## 8. Class Diagram and Interface Specification

### a. Class Diagram

The following section contains the class diagram for the Digitized Library Management System, including class names, relationships, attributes, and operations.



**b. Data Types and Operation Signatures**

**+ User Class**

- **Attributes:**
    - userID: int - unique identifier for the user
    - username: string - username chosen by the user
    - passwordHash: string - hashed password for security
    - email: string - email address of the user
    - registrationDate: datetime - date and time when user registered
    - address: string - address of the user
- **Operations:**
    - +getUserID(): int - retrieves user's ID
    - +getUsername(): string - retrieves username
    - +verifyPassword(password: string): bool - verifies if the provided password matches the stored hash
    - +getEmail(): string - retrieves email
    - +getRegistrationDate(): datetime - retrieves registration date
    - +updateProfile(username: string, email: string, address: string): void - updates user profile information

**+ Book Class**

- **Attributes:**
    - bookID: int - unique identifier for the book
    - title: string - title of the book
    - ISBN: string - International Standard Book Number
    - publicationYear: int - year the book was published
    - availability: bool - indicates if the book is available for loan
- **Operations:**
    - +getBookID(): int - retrieves book ID
    - +getTitle(): string - retrieves book title
    - +getISBN(): string - retrieves ISBN
    - +getPublicationYear(): int - retrieves publication year
    - +isAvailable(): bool - checks book availability
    - +updateAvailability(availability: bool): void - updates book availability

**+ Author Class**

- **Attributes:**
    - authorID: int - unique identifier for the author
    - name: string - name of the author
    - biography: string - biography of the author
- **Operations:**
    - +getAuthorID(): int - retrieves author ID
    - +getName(): string - retrieves author name
    - +getBiography(): string - retrieves biography of the author

+ **Genre Class**

- **Attributes:**
    - genreID: int - unique identifier for the genre
    - name: string - name of the genre
    - description: string - description of the genre
- **Operations:**
    - +getGenreID(): int - retrieves genre ID
    - +getName(): string - retrieves genre name
    - +getDescription(): string - retrieves description of the genre

+ **Loan Class**

- **Attributes:**
    - loanID: int - unique identifier for the loan
    - bookID: int - identifier of the loaned book
    - userID: int - identifier of the user who borrowed the book
    - loanDate: datetime - date and time when the book was loaned
    - dueDate: datetime - date and time when the book is due
    - returnDate: datetime - date and time when the book was returned
- **Operations:**
    - +getLoanID(): int - retrieves loan ID
    - +getBookID(): int - retrieves book ID
    - +getUserID(): int - retrieves user ID
    - +getLoanDate(): datetime - retrieves loan date
    - +getDueDate(): datetime - retrieves due date
    - +getReturnDate(): datetime - retrieves return date
    - +updateReturnDate(returnDate: datetime): void - updates return date

+ **Reservation Class**

- **Attributes:**
    - reservationID: int - unique identifier for the reservation
    - bookID: int - identifier of the reserved book
    - userID: int - identifier of the user who reserved the book
    - reservationDate: datetime - date and time when the reservation was made
    - status: string - current status of the reservation (e.g., pending, completed)
- **Operations:**
    - +getReservationID(): int - retrieves reservation ID
    - +getBookID(): int - retrieves book ID
    - +getUserID(): int - retrieves user ID
    - +getReservationDate(): datetime - retrieves reservation date
    - +getStatus(): string - retrieves reservation status
    - +updateStatus(status: string): void - updates reservation status

+ **Review Class**

- **Attributes:**
    - reviewID: int - unique identifier for the review
    - bookID: int - identifier of the reviewed book
    - userID: int - identifier of the user who wrote the review
    - rating: int - rating for the book (e.g., from 1 to 5)
    - comment: string - user comment for the book
    - reviewDate: datetime - date and time when the review was written
- **Operations:**
    - +getReviewID(): int - retrieves review ID
    - +getBookID(): int - retrieves book ID
    - +getUserID(): int - retrieves user ID
    - +getRating(): int - retrieves rating for the book
    - +getComment(): string - retrieves comment for the book
    - +getReviewDate(): datetime - retrieves review date

+ **Library Class**

- **Attributes:**
    - libraryID: int - unique identifier for the library
    - name: string - name of the library
    - address: string - address of the library

- **Operations:**
    - +getLibraryID(): int - retrieves library ID
    - +getName(): string - retrieves library name
    - +getAddress(): string - retrieves library address

**+ Librarian Class**

- **Attributes:**
    - librarianID: int - unique identifier for the librarian
    - userID: int - identifier of the librarian's user account
    - libraryID: int - identifier of the library the librarian works for
    - hireDate: datetime - date the librarian was hired
- **Operations:**
    - +getLibrarianID(): int - retrieves librarian ID
    - +getUserID(): int - retrieves user ID
    - +getLibraryID(): int - retrieves library ID
    - +getHireDate(): datetime - retrieves hire date

**+ Notification Class**

- **Attributes:**
    - notificationID: int - unique identifier for the notification
    - userID: int - identifier of the user receiving the notification
    - message: string - message content of the notification
    - timestamp: datetime - date and time when the notification was sent
- **Operations:**
    - +getNotificationID(): int - retrieves notification ID
    - +getUserID(): int - retrieves user ID
    - +getMessage(): string - retrieves message content
    - +getTimestamp(): datetime - retrieves timestamp

### c. Design Patterns

| Classes | User | Book | Author | Genre | Loan | Reservation | Review | Library | Librarian | Notification |
|---|---|---|---|---|---|---|---|---|---|---|
| User | X | | | | | X | X | | | X |
| Book | | X | X | X | X | X | X | X | | X |
| Author | | X | X | | | | | | | |
| Genre | | X | | X | | | | | | |
| Loan | X | X | | | X | | | | X | X |
| Reservation | X | X | | | | X | | | X | X |
| Review | X | X | | | | | X | | | |
| Library | | X | | | | | | X | X | X |
| Librarian | X | | | | X | X | | X | X | X |
| Notification | X | X | | | X | X | | X | X | X |
| NotificationPublisher | | | | | | | | | | X |
| EmailModule | X | | | | | | | | | X |

### d. Object Constraint Language (OCL)

**Library Class**

context Library::getLibraryID() post:
this.#libraryID returned

context Library::getName() post:
this.#name returned

context Library::getAddress() post:
this.#address returned

**Book Class**

context Book::getBookID() post:
this.#bookID returned

context Book::getTitle() post:
this.#title returned

context Book::getISBN() post:
this.#ISBN returned

context Book::getPublicationYear() post:
this.#publicationYear returned

context Book::isAvailable() post:
this.#availability returned

context Book::updateAvailability(availability: Boolean) pre:
availability <> null
context Book::updateAvailability(availability: Boolean) post:
this.#availability updated

**Author Class**

context Author::getAuthorID() post:
this.#authorID returned

context Author::getName() post:
this.#name returned

context Author::getBiography() post:
this.#biography returned

**Genre Class**

context Genre::getGenreID() post:
this.#genreID returned

context Genre::getName() post:
this.#name returned

context Genre::getDescription() post:
this.#description returned

**Review class**

context Review::getReviewID(): int post:
this.#reviewID returned

context Review::getBookID(): int post:
this.#bookID returned

context Review::getUserID(): int post:
this.#userID returned

context Review::getRating(): int post:
this.#rating returned

context Review::getComment(): string post:
this.#comment returned

context Review::getReviewDate(): datetime post:
this.#reviewDate returned

**User Class**

context User::getUserID(): int post:
this.#userID returned

context User::getUsername(): string post:
this.#username returned

context User::verifyPassword(password: string): bool post:
result = (self.passwordHash = password.hash)

context User::getEmail(): string post:
this.#email returned

context User::getRegistrationDate(): datetime post:
this.#registrationDate returned

context User::updateProfile(username: string, email: string, address: string): void post:

self.#username = username and self.#email = email and self.#address = address

**Librarian Class**

context Librarian::getLibrarianID(): int post:
this.#librarianID returned

context Librarian::getUserID(): int post:
this.#userID returned

context Librarian::getLibraryID(): int post:
this.#libraryID returned

context Librarian::getHireDate(): datetime post:
this.#hireDate returned

**Reservation Class**

context Reservation::getReservationID(): int post:
this.#reservationID returned

context Reservation::getBookID(): int post:
this.#bookID returned

context Reservation::getUserID(): int post:
this.#userID returned

context Reservation::getReservationDate(): datetime post:
this.#reservationDate returned

context Reservation::getStatus(): string post:
this.#status returned

context Reservation::updateStatus(status: string): void post:
this.#status = status

**Notification Class**

context Notification::getNotificationID(): int post:
this.#notificationID returned

context Notification::getUserID(): int post:
this.#userID returned

context Notification::getMessage(): string post:
this.#message returned

context Notification::getTimestamp(): datetime post:
this.#timestamp returned

**Loan Class**

context Loan::getLoanID(): int post:
this.#loanID returned

context Loan::getBookID(): int post:
this.#bookID returned

context Loan::getUserID(): int post:
this.#userID returned

context Loan::getLoanDate(): datetime post:
this.#loanDate returned

context Loan::getDueDate(): datetime post:
this.#dueDate returned

context Loan::getReturnDate(): datetime post:
this.#returnDate returned

context Loan::updateReturnDate(returnDate: datetime): void post:
this.#returnDate = returnDate

==========

Report #2 - Full report

**9. System Architecture and System Design**

**a. Identifying Subsystems**

**b. Architecture Styles**
**c. Mapping Subsystem to Hardware**
**d. Connectors and Network Protocols**
**e. Global Control Flow**
**f. Hardware Requirements**

**10. Algorithms and Data Structures**
**a. Data Structures**
**b. Concurrency**
**11. User Interface Design and Implementation**
**12. Design of Tests**
**a. Test Cases**
**b. Test Coverage**
**c. Solution**
**d. Conclusion**
**13. Reflective Essay and Peer Evaluation**

**14. History of Work, Current Status, and Future Work**
**a. History of Work**
**b. Current Status**
**c. Future Work**

==========

**Plan of work**

*Tools:* Java, JavaScript, PHP, SQL, HTML, CSS
*Successful System:* A successful Library Management System would mean:

- Clients can filter search results by author, genre, or reading level without confusion.
- Checking books in or out is straightforward and updates availability in real time.
- Users have their own accounts to track reading histories, wish lists, and current loans.
- The database accurately stores all transactions & information without data loss or errors.
- Allow library owners to digitally manage their inventory.

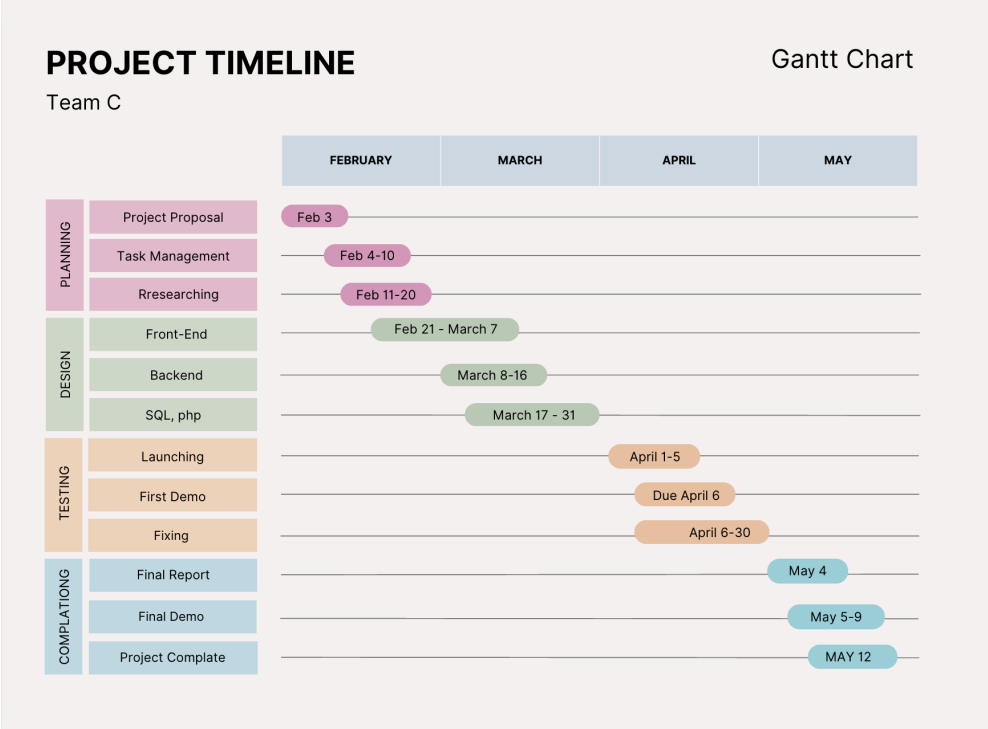
**Project Management**

Project Timeline:

- **Weeks 1–2:** Gather requirements, set up environment, and begin database schema design.

- **Weeks 3–4:** Implement basic user authentication and simple search.
- **Weeks 5–6:** Finalize check-in/check-out functionality, integrate availability tracking.
- **Weeks 7–8:** Develop the user account features (wish lists, history).
- **Weeks 9–10:** Refine the interface, conduct testing with sample data.
- **Weeks 11–12:** Make any necessary changes, finalize documentation, prepare for demonstration.

## Outline of Scheduled Plan of Work




[Gantt Chart Canva Link]

**References**

Alfa eBooks. (*n.d.*). *Electronic Library Management System*. Retrieved from https://www.alfaebooks.com/electronic_library_management_system

Mastersoft. (*n.d.*). *Library Management System*. Retrieved from https://www.iitms.co.in/library-management-system/

Naira Project. (*n.d.*). *Electronic Library Management System*. Retrieved from https://nairaproject.com/projects/2212.html