

# Library Management System

## ***Presented By:***

Mazen Hussein Elemam  
23016099

Mohamed Omar Sayed  
23013434

Mohab Ayman Abdullah  
23013662

Abdulrahman Ahmed Mahmoud  
23017568

## ***Supervised By:***

**Dr. Ayman Hasanain**

## ***Co-Supervision:***

**Eng. Ahmed Abdelmonem**  
**Eng. Mohammed Abdelmonem**  
**Eng. Nada Hamdy**

# 1.Introduction

The Library Management System (LMS) is a sophisticated, enterprise-grade software architecture designed to fundamentally transform the operational efficiency of modern library environments. By transitioning from disparate manual ledgers to a unified, centralized database, the system eradicates human error and redundancy, creating a seamless digital ecosystem for its three primary stakeholders: **Members, Librarians, and Administrators.**

The system provides holistic management of the library's inventory lifecycle, tracking every asset from its initial **acquisition and cataloging** (entry of ISBN, title, and author data) to its active **circulation** (real-time availability status, borrowing, and reservation), and finally to its **maintenance or archiving**. Beyond basic inventory tracking, the LMS serves as an intelligent enforcement engine for library policies. It automates complex logic such as **fine calculation** for overdue items, **eligibility validation** (blocking users with outstanding debts), and **asset preservation** (recording book condition and assessing damage fees). This ensures a secure, transparent, and highly efficient environment where resources are optimized, and administrative overhead is significantly reduced.

## 2.Problem Definition

The system is designed to resolve specific critical failures inherent in manual library administration:

- **Operational Bottlenecks:** Manual processing of book loans and returns is slow, creating queues and reducing Librarian productivity.
- **Inventory Inaccuracy:** Without real-time tracking, books are often listed as "available" when they are lost, misplaced, or damaged.
- **Revenue Leakage:** Manual fine calculation is error prone. Staff may miss overdue days or fail to assess damage fees consistently.
- **Data Vulnerability:** Physical ledgers cannot be easily backed up. The digital system introduces `backupDatabase()` and `archiveOldRecords()` capabilities to ensure data longevity.
- **Lack of Accountability:** Manual systems struggle to track who processed a transaction. This system introduces distinct login sessions for Librarians and Admins to enforce accountability.

## 3. System Description & Architecture

The system is defined by its data structures (Class Diagram) and interaction flows (Sequence/Activity Diagrams).

### 3.1 Data Actors & Entities

- **The System (Core Logic):** Acts as an autonomous entity that performs background tasks such as `calculateFines()`, `sendNotifications()`, and `updateBookAvailability()`. It serves as the bridge between the user interface and the database.
- **Member:** The end-user who interacts with the system to `searchBooks()`, `borrowBook()`, `returnBook()`, and `payFine()`.
- **Librarian:** The operational staff responsible for the catalog (`addNewBook`, `removeBook`), circulation control (`acceptBookReturn`, `calculateFine`), and user oversight (`blockMember`).
- **Admin:** The system controller with high-level permissions to `manageLibrarians()` and `configureSystemSettings()`.
- **Book:** A central entity containing attributes like ISBN, Title, and `isAvailable`. It tracks its own state, allowing the system to toggle availability instantly.
- **Borrow:** A transaction record that links a **Member** to a **Book**. It stores critical dates (`issueDate`, `dueDate`, `returnDate`) and contains logic to `calculateOverdueDays()`.

### 3.2 Key Process Flows

- **The Borrowing Lifecycle:** A multi-step validation process where the Member searches for a book      System checks availability  
Member      submits a request      System verifies member status  
(e.g., no unpaid fines)      Admin performs a final review      Borrow  
record is created.
- **The Return Lifecycle:** Includes a physical inspection step. The Librarian checks the book's condition. If damaged, a "Damage Fee" is added to the standard overdue fine (if any) before the book is marked as available again.

## 4. User Requirements

The system must satisfy the unique needs of its three user groups as depicted in the Use Case diagrams:

### For Members

- **Self-Service Portal:** Members must be able to log in to a personal dashboard to view their borrowing history (viewBorrowedHistory) and active loan status.
- **Catalog Interaction:** The ability to execute complex searches (by Title, Author, Category) and view detailed book status (Available/Unavailable).
- **Reservation Management:** Members need the ability to reserveBook() if an item is currently checked out and cancelReservation() if no longer needed.

### For Librarians

- **Inventory Control:** Tools to enter new book data into the database and generate physical barcodes for tracking.
- **Member Management:** Capability to verify member IDs during physical visits and execute a blockMember() command for rule violations.

- **Fine Processing:** A dedicated interface to calculate total fines (Overdue + Damage) and mark them as "Paid" in the system.

#### For Administrators

- **Staff Oversight:** A secure interface to add, update, or remove Librarian accounts and assign permissions.
- **Strategic Reporting:** The ability to generateReports() to analyze system usage, financial intake from fines, and inventory status.

## 5. System Requirements

### 5.1 Functional Requirements (FRs)

These define the specific behaviors the system is programmed to execute:

1. **Automated Availability Logic:** When a book is borrowed, the system must automatically set its `isAvailable` attribute to false. Upon return, it must reset to true only *after* the fine/damage check is cleared.
2. **Fine Calculation Engine:**
  - The system must compare `returnDate` against `dueDate`.
  - *Logic:* If (`returnDate` > `dueDate`) THEN `Fine` = (`Days_Overdue` \* `Daily_Rate`).
  - *Logic:* If (`Book_Condition` == `Damaged`) THEN `Total_Fine` = `Fine` + `Damage_Fee`.
3. **Search & Retrieval:** The system must return "N/A" or "Not Found" messages if a search query yields no results, preventing users from requesting non-existent books.
4. **Notifications:** The system must trigger alerts for specific events: "Book Approved," "Request Rejected" (with reason), and "Fine Details."
5. **Database Maintenance:** The system must provide functions to `archiveOldRecords()` to keep the active database performant and `backupDatabase()` to prevent data loss.

## 5.2 Non-Functional Requirements (NFRs)

These define the quality attributes of the system:

### 1. **Security & Access Control:**

- **Authentication:** All users (Librarian, Admin) must authenticate via a login mechanism.
- **Authorization:** The system must enforce Role-Based Access Control (RBAC). For example, a Member cannot access the addNewBook function; only a Librarian can.

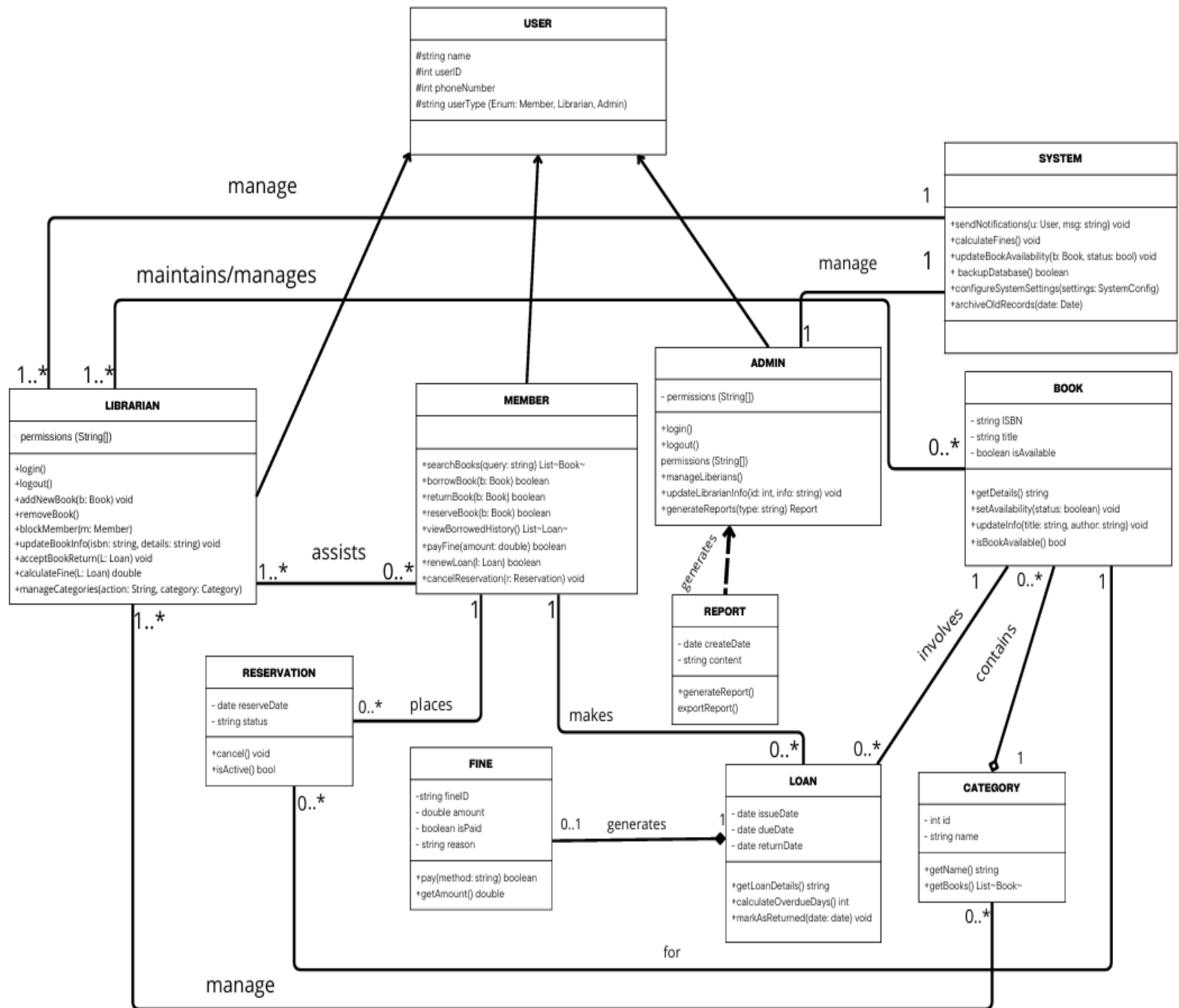
2. **Data Integrity:** The system must ensure that a Book cannot be deleted if it is currently linked to an active Loan record.

3. **Performance:** The "Search" function must be optimized to handle queries against a large database of books without significant latency.

4. **Scalability:** The architecture (separating the Database from the System logic) supports adding more books and members without rewriting the core application code.

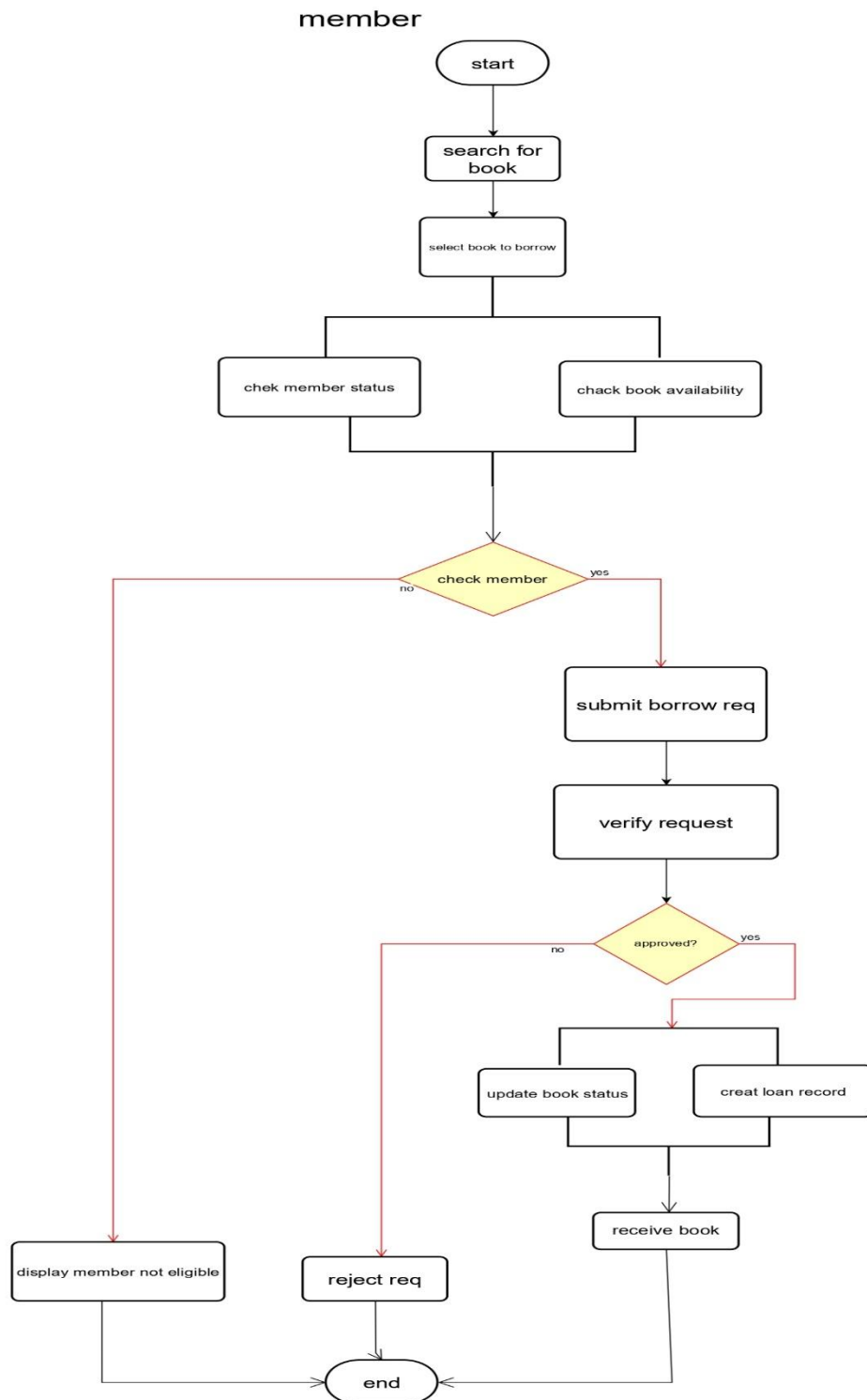
5. **Usability:** The system must provide clear feedback messages (e.g., "Member Not Eligible," "Book Successfully Issued") to ensure users understand the outcome of their actions.

# CLASS DIAGRAM



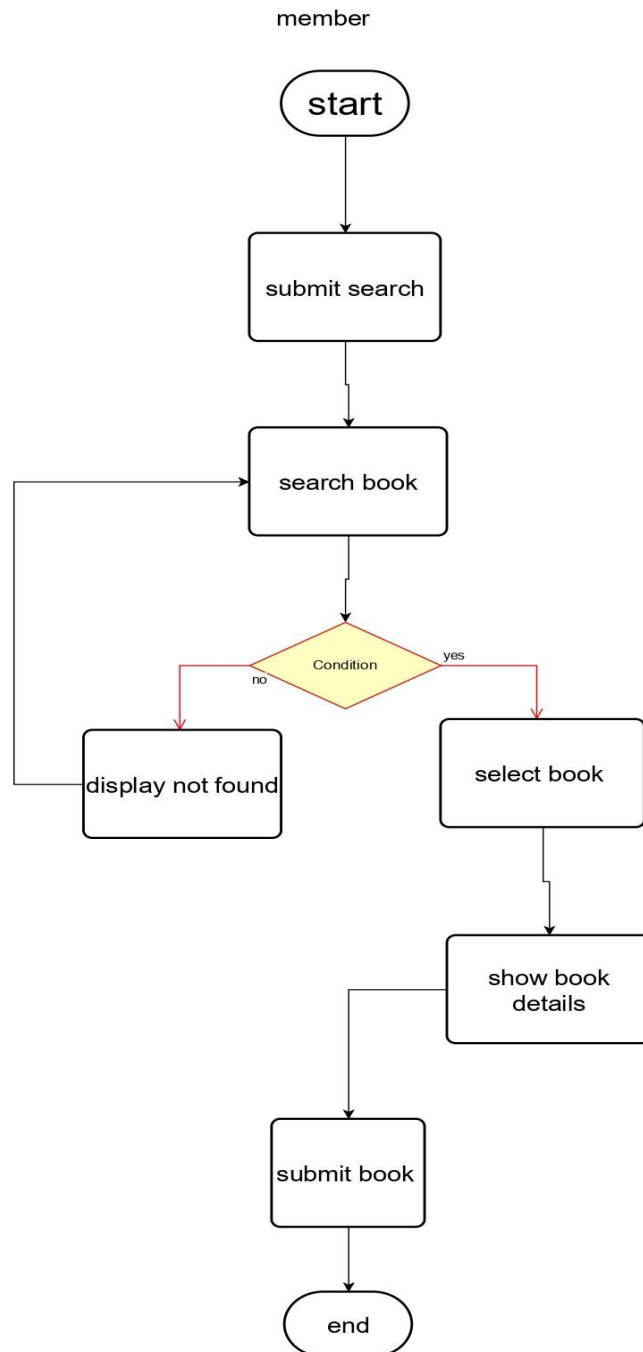
# ACTIVITY DIAGRAM

## 1. Borrow Function



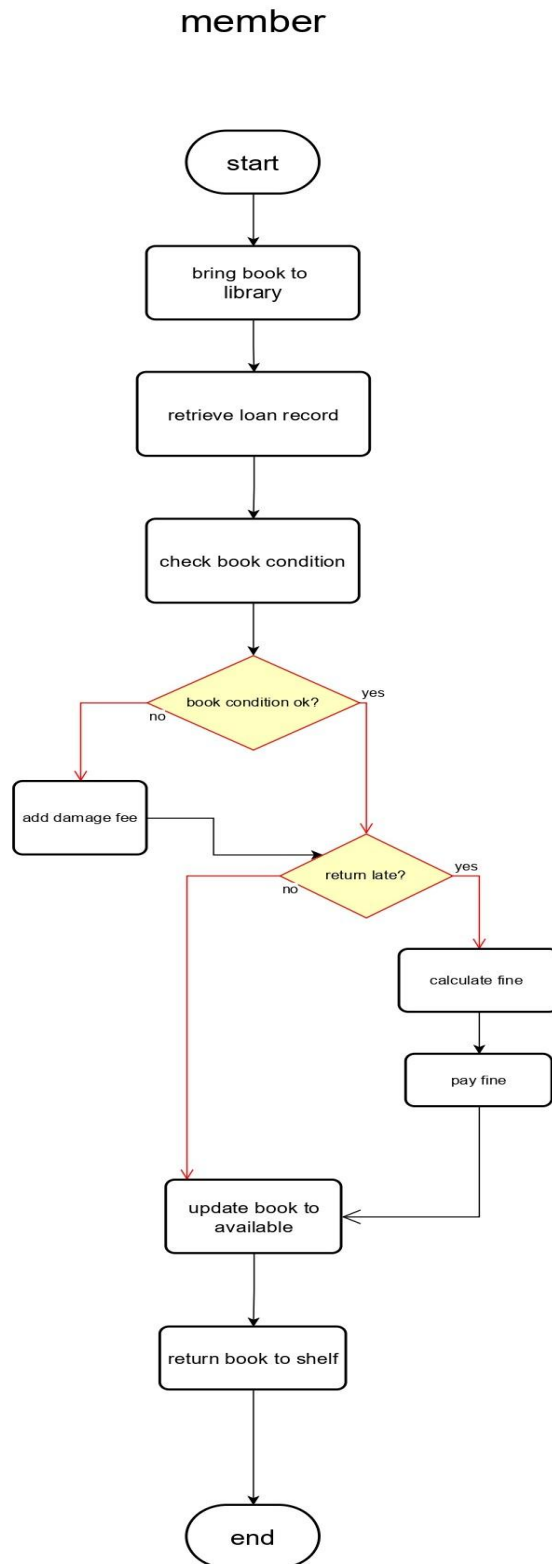
# ACTIVITY DIAGRAM

## 2.Search Function



# ACTIVITY DIAGRAM

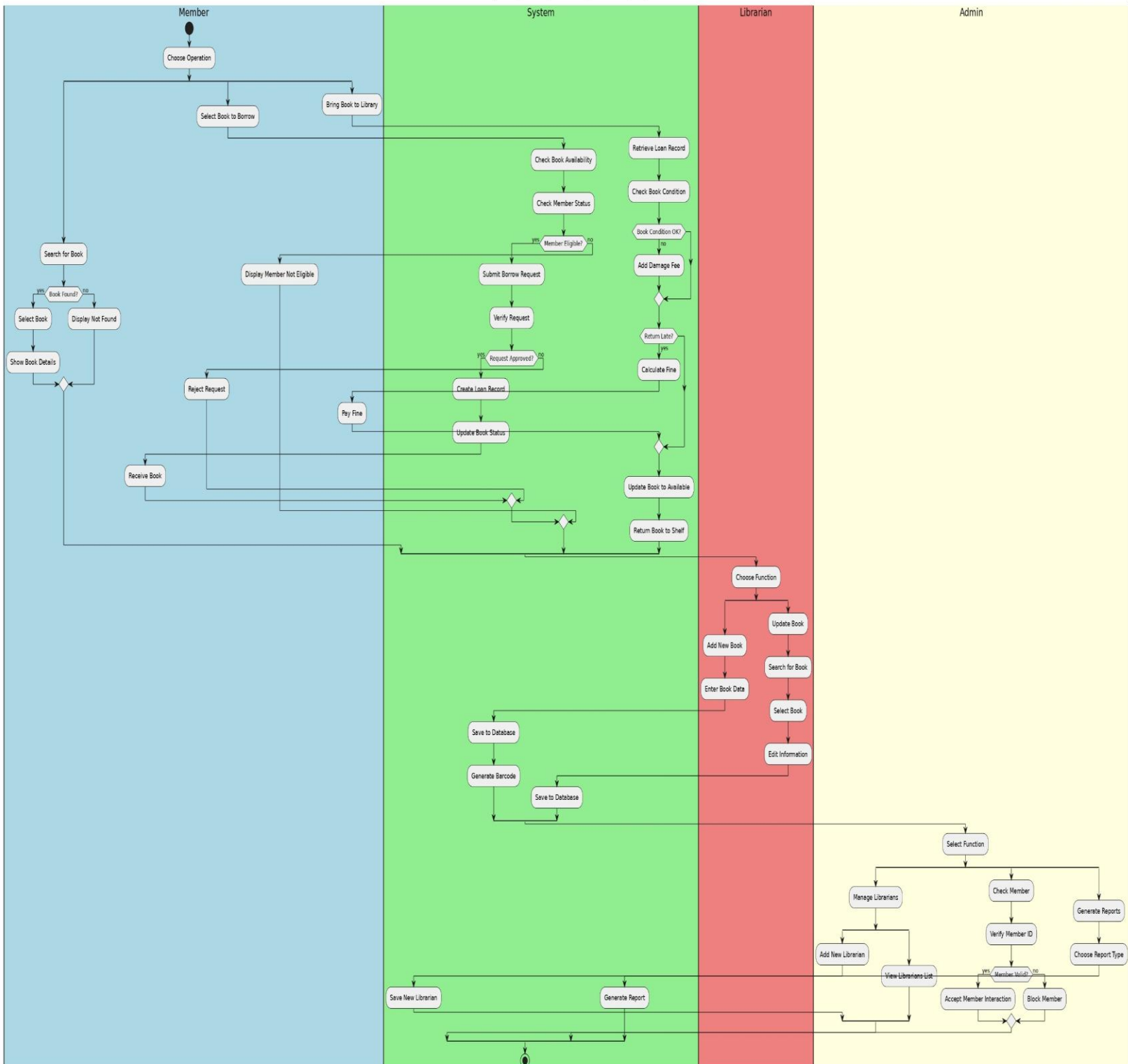
## 3. Return Function



# ACTIVITY DIAGRAM

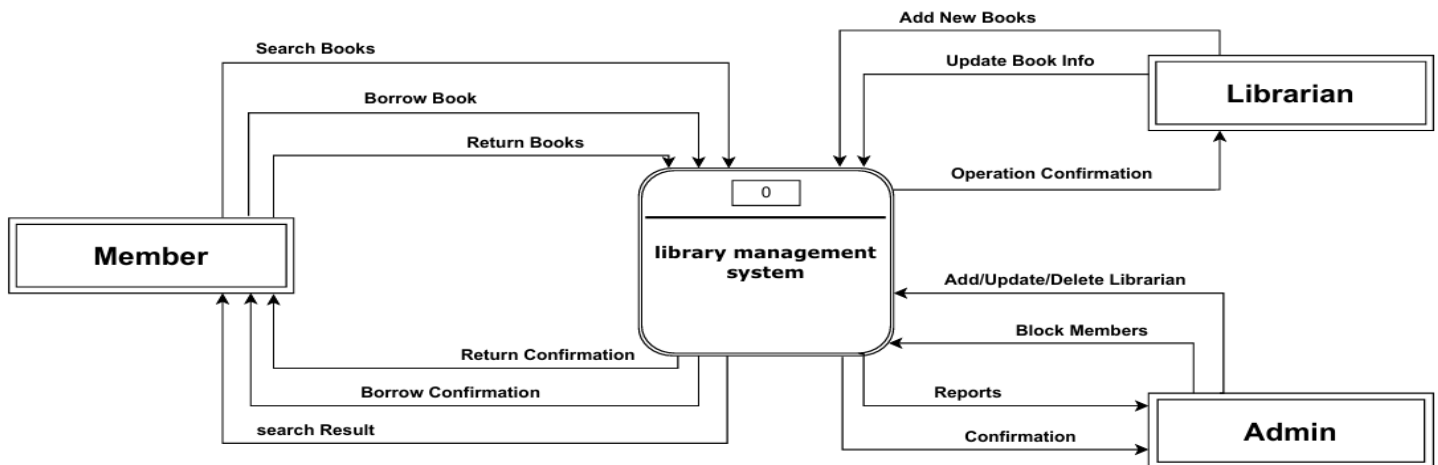
## Final Diagram

Library Management System - Complete Activity Diagram

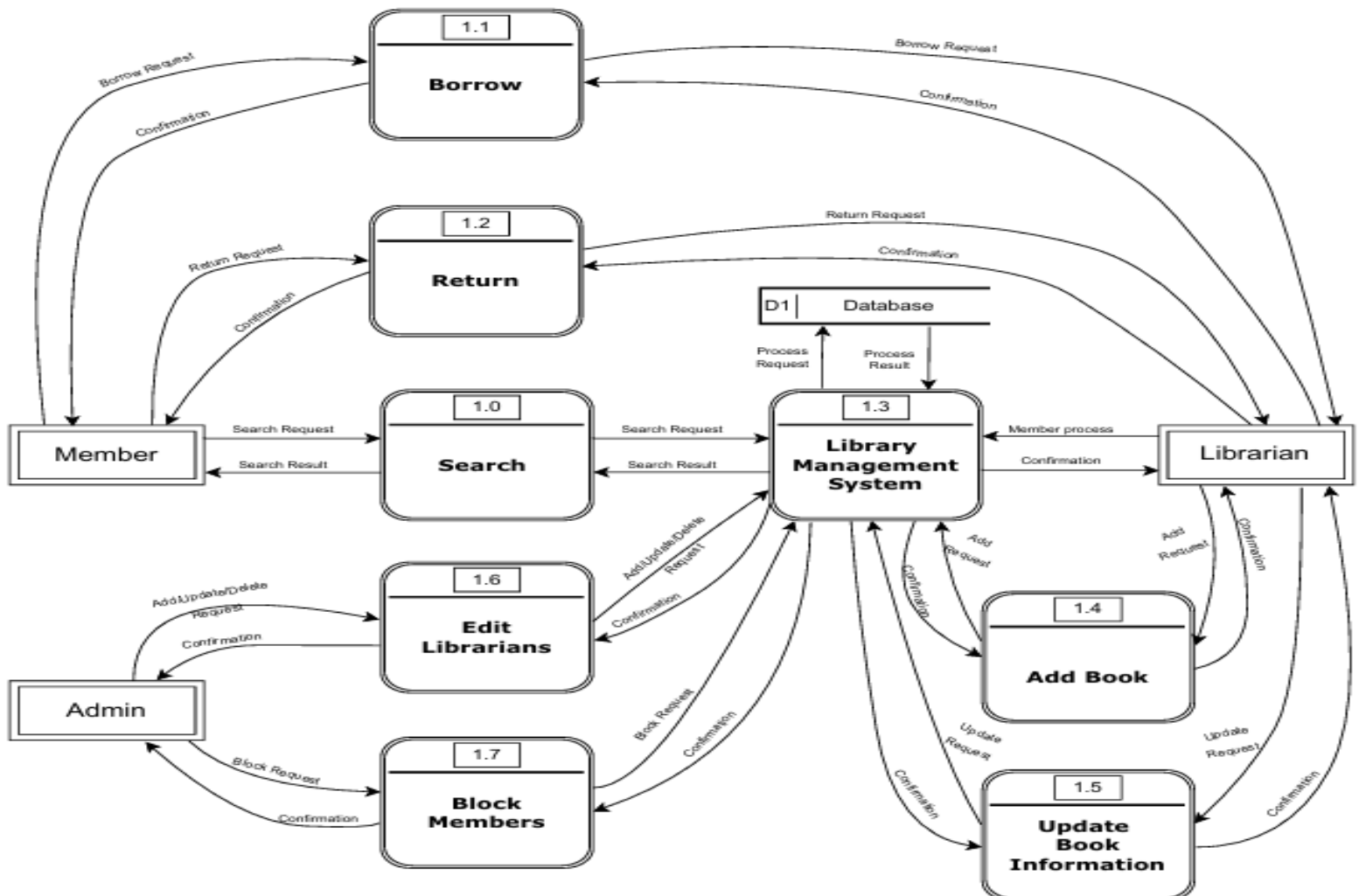


# DATA FLOW DIAGRAM

## Level 0

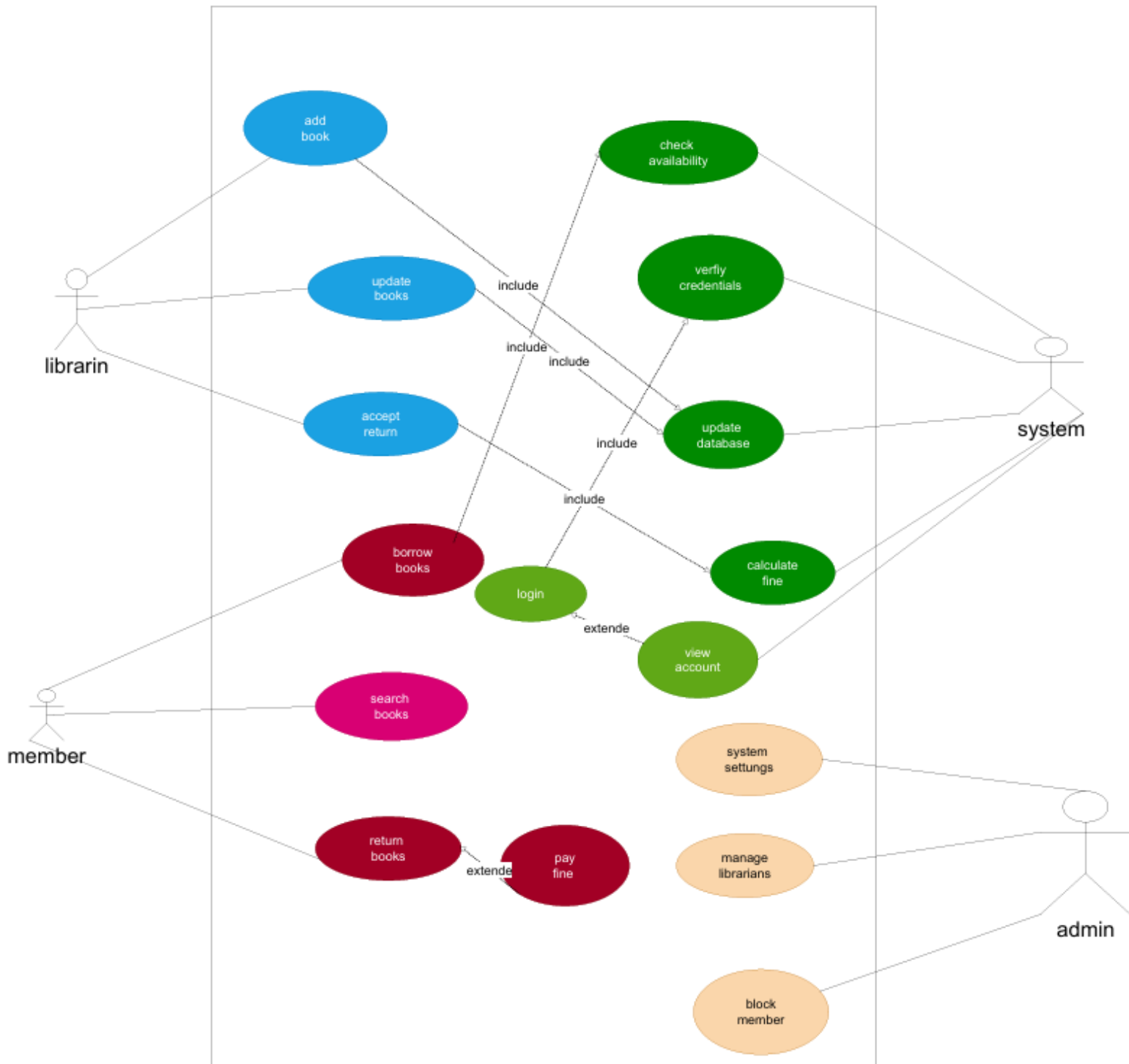


## Level 1



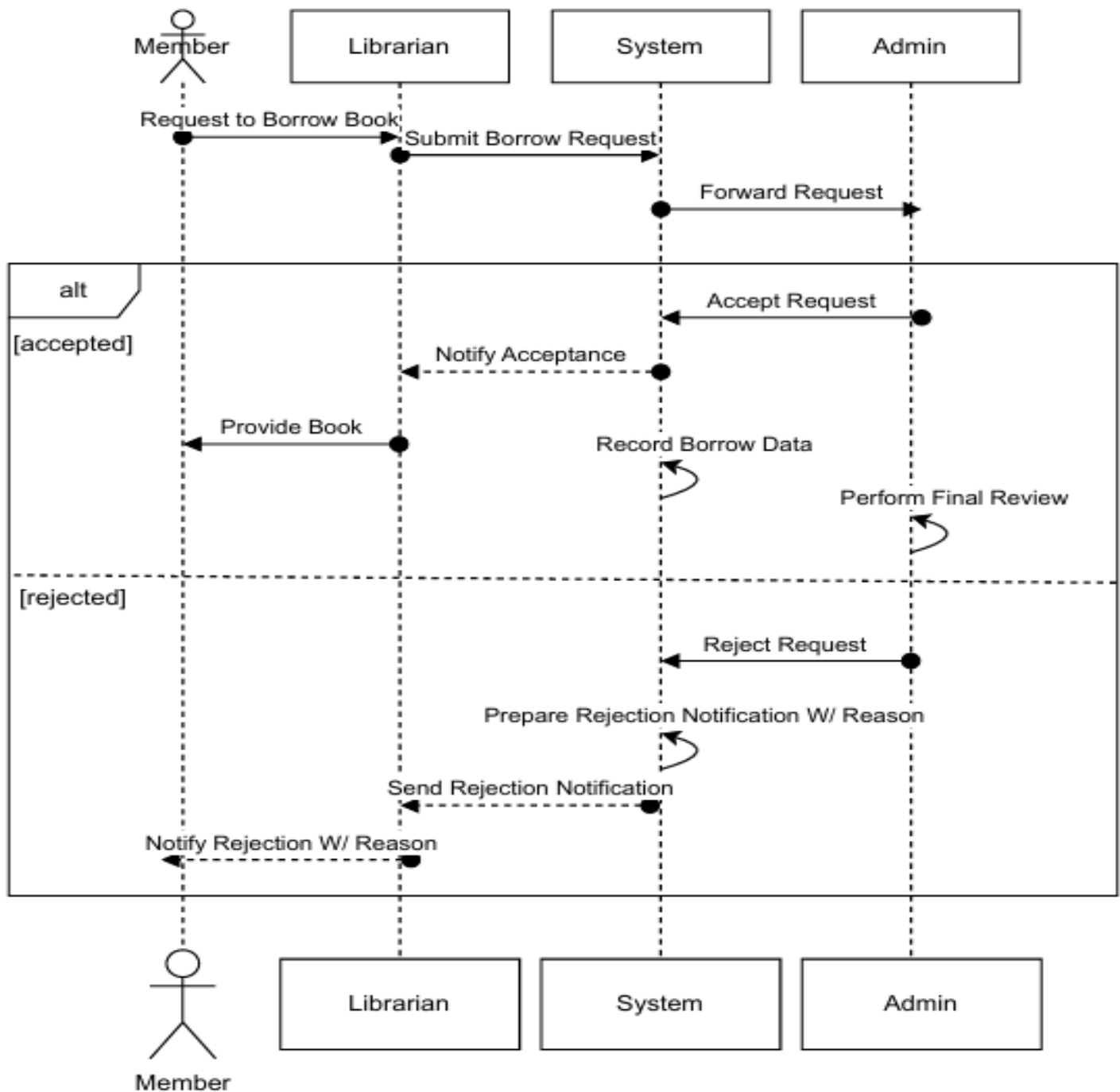
# USE CASE DIAGRAM

## Library System



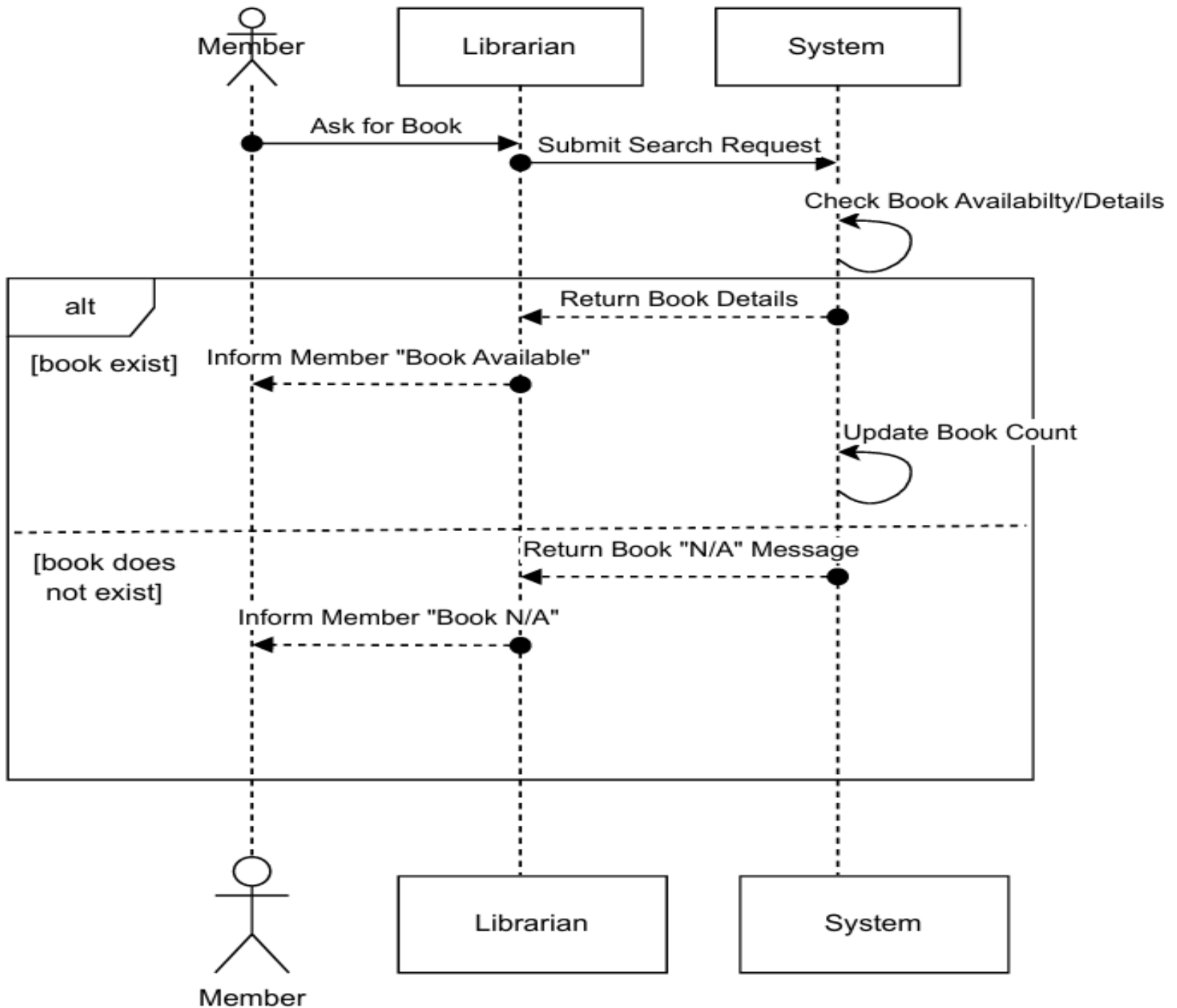
# SEQUENCE DIAGRAM

## 1.Borrow Function



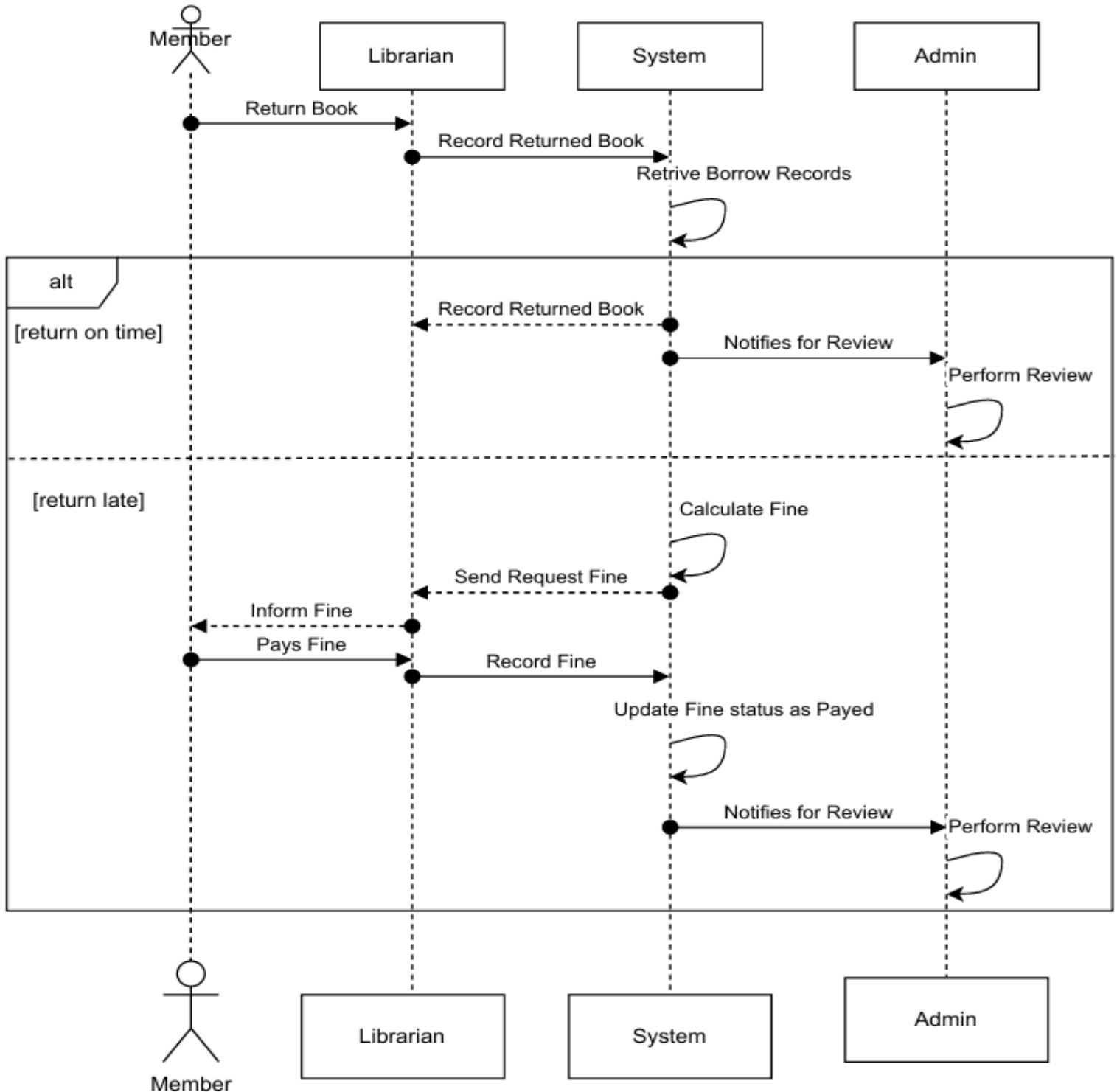
# SEQUENCE DIAGRAM

## 2.Search Function



# SEQUENCE DIAGRAM

## 3.Return Function



# SEQUENCE DIAGRAM

## Final Diagram

