SRS

Software Requirements Specification

Project name:

Library Management System

1. Introduction

1.1 Purpose

This SRS document describes the functional and non-functional requirements for the Library Management System (LMS). The system will be built using React for the frontend and Spring Boot for the backend.

1.2 Scope

The LMS will allow administrators, librarians, and members to manage library operations including user management, book cataloging, borrowing/returning, fine calculation, and reporting.

In scope	Out of scope
User Management	Inventory management or external book ordering
Book Management	Advanced analytics or AI-based book recommendations
Borrowing & Returning	Integration with payment or financial services
Reports generation	
Authentication, Registeration & Security	

1.3 Definitions

List and explain any terms used in this document.

API	Application Programming Interface – a set of rules that allows different software components to communicate.
LMS:	Library Management System
JWT:	JSON Web Token (authentication)
AOP:	Aspect Oriented Programming
Admin:	System administration(Highest Privilege)
Librarian:	Book handling, borrow request approval or rejection
Member:	Common user who sends borrow requests for books (lowest privilege)

2. Overall description

2.1 Product perspective

The LMS is a web-based system following micro-service architecture. React handles the UI and communicates with Spring Boot APIs for all operations.

2.2 Product features

LibraryManagementSystem(LMS) will provide the following major features:

Authentication: Secure logging-in using JWT with B-Crypt hashing for passwords.

Borrowing & Returning: Librarians create & manage books. The system tracks borrowing details, due dates, and Blocks users for overdue books after 30 days.

User Management (Admin): Admin can edit, delete, and view users.

Reports: Generate reports on borrowed books, borrowing users and their current borrowing status

Book Management: Admin and Librarian can manage books (add, edit, delete, view). Users can search a books.

2.3 System context

- API Gateway routes to *Authentication and Library services*; MongoDB is used as the database in a containerized (dockerized) environment.

2.4 User classifications, roles and characteristics

Identify the various user classes that you anticipate will use this product, and their role in the system.

User class	Description	Experience level	Frequency of use
Admin users	Manages users, books, and generates & view reports. Oversees most system operations.	Advanced	High
Librarian	Issues and returns books, updates and adds book records.	Intermediate	High
Member/user	Searches for books, borrows and returns books(with limited behavior), views borrowing history.	Basic	Low

3. Functional requirements

Use this section for the system's functional requirements. Describe features, behaviors, and interactions that fulfill user needs or business rules.

3.1 Authentication

- The system shall allow users to log in using Username and Password. The backend shall use JWT for secure authentication.
- Bcrypt shall be used for password hashing
- Unauthorized users shall not access protected pages.

3.2 User Management (Admin)

- The system shall allow admin access on book creation and management.
- The system shall allow the Admin to edit user privileges. The system shall allow the Admin to edit user data. The system shall allow the Admin to delete users.

3.3 Book Management

- The system shall allow Admin/Librarian to add new books. The system shall allow editing and deleting books. The system shall store book details (title, author, ISBN, copies).
- The system shall allow users to search for books.

3.4 Borrowing & Returning

The Librarian shall approve of borrow requests to members.

- he Librarian shall be allowed rejection of borrow requests to members.
- The system shall check book availability before issuing.
- The system shall record borrowing details, due dates, and related data for reports and analytics.

3.5 Reports

- The system shall generate reports including:
- Borrowed books.
- Amount of borrowed books
- Amount of borrowing users
- Get borrow history per user

Book availability

4. External interface requirements

4.1 User interfaces	Web UI: Implemented using React.js. Supports modern browsers (Chrome, Edge, Firefox). Forms: For login, user registration, book management, borrowing, and returns. Dashboard: Admin Librarian, member dashboard for managing books, users, reports. and making a borrow request
4.2 Hardware interfaces	Standard PC or laptop with internet connection. Docker installed.
4.3 Software interfaces	Backend API: REST APIs implemented in Spring Boot. Database: MongoDB + Docker for data storage. and persistence Authentication: Spring Security + JWT + Bcrypt.
4.4 Communications interfaces	HTTP/HTTPS for frontend-backend communication.

5. Non-functional requirements

5.1 Performance	The system shall respond to user requests within 5 seconds. The system shall support at least 500 concurrent users.
5.2Security	Passwords shall be encrypted using BCrypt. JWT shall be used for authentication. Role-based access control shall be enforced.
5.3 Usability	The interface shall be user-friendly and responsive. Error messages shall be clear and helpful.
5.4 Reliability	System uptime shall be at least 99%.
5.5 Maintainability	The system shall follow microservice w/modular architecture. APIs shall follow REST conventions.

6.Architecture and components

6.1 Authentication service and Library service shall be implemented as two separate microservices.

6.2 API Gateway shall be used (Spring Cloud)

6.3 Authentication service for Adding and authenticating users and implementing JWT.

6.4 Library services modules being books, borrow, reports & userManagement.

6.5 A single dockerized container of MongoDb.

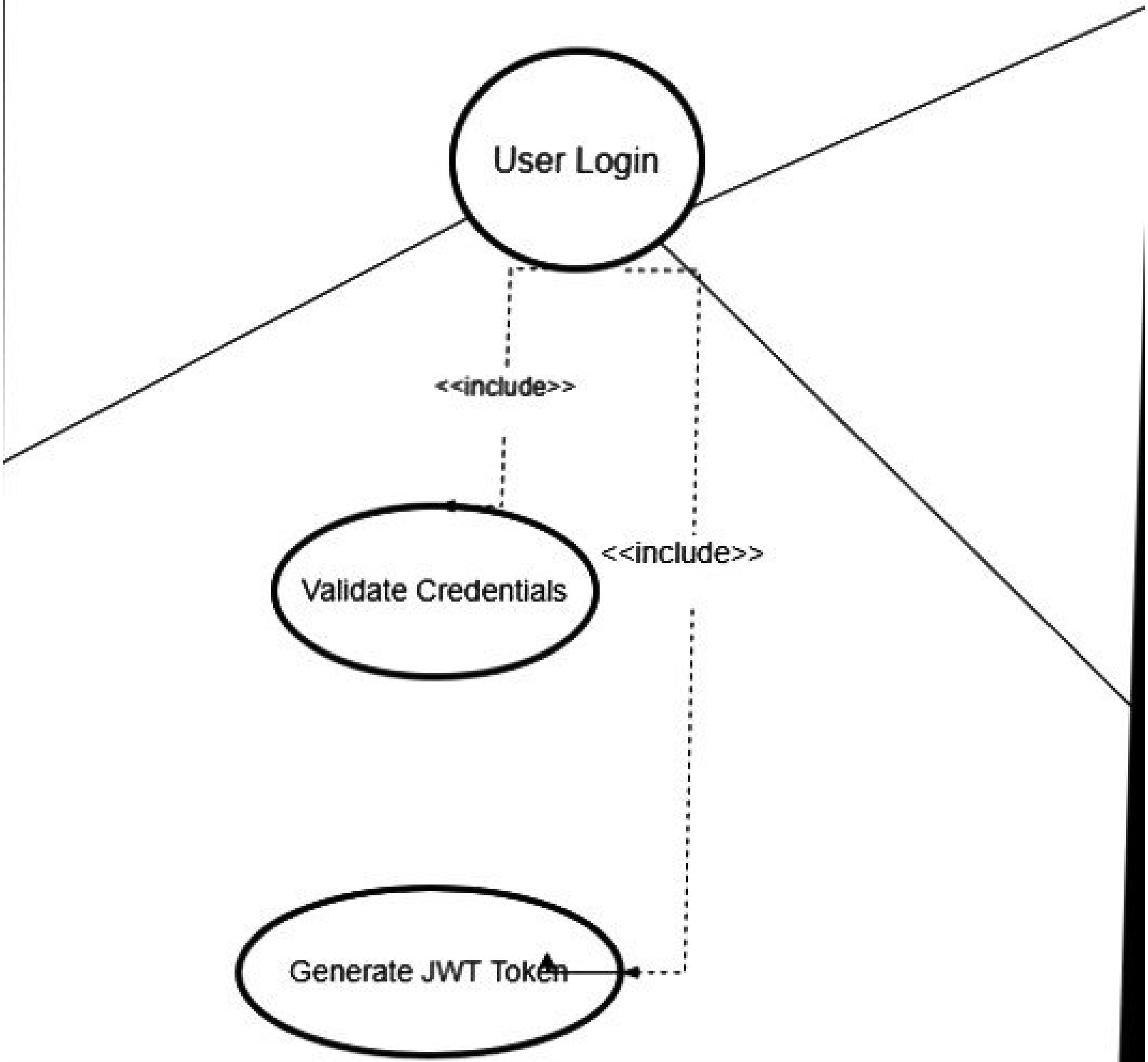
7.Database schema

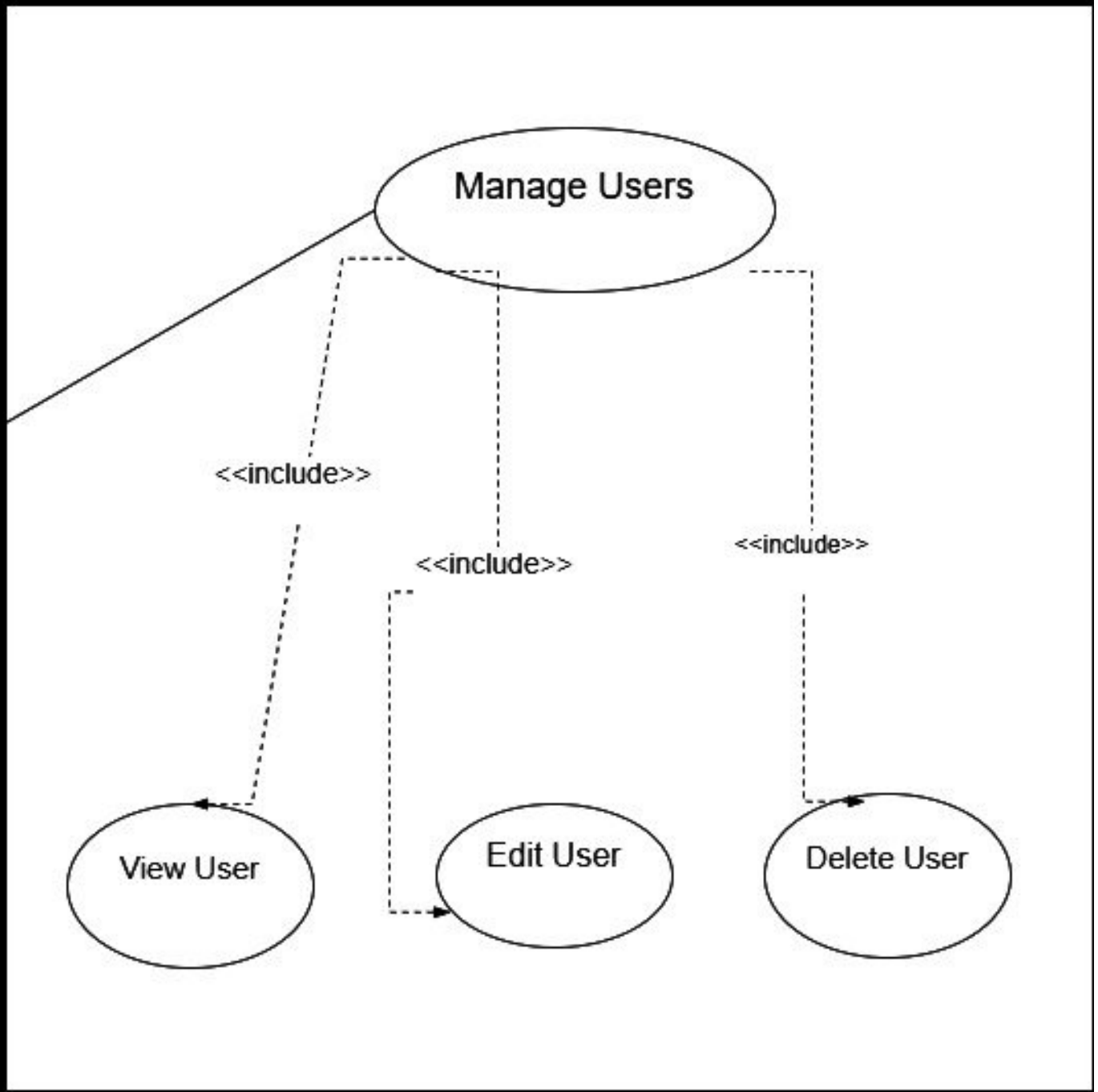
Collections:

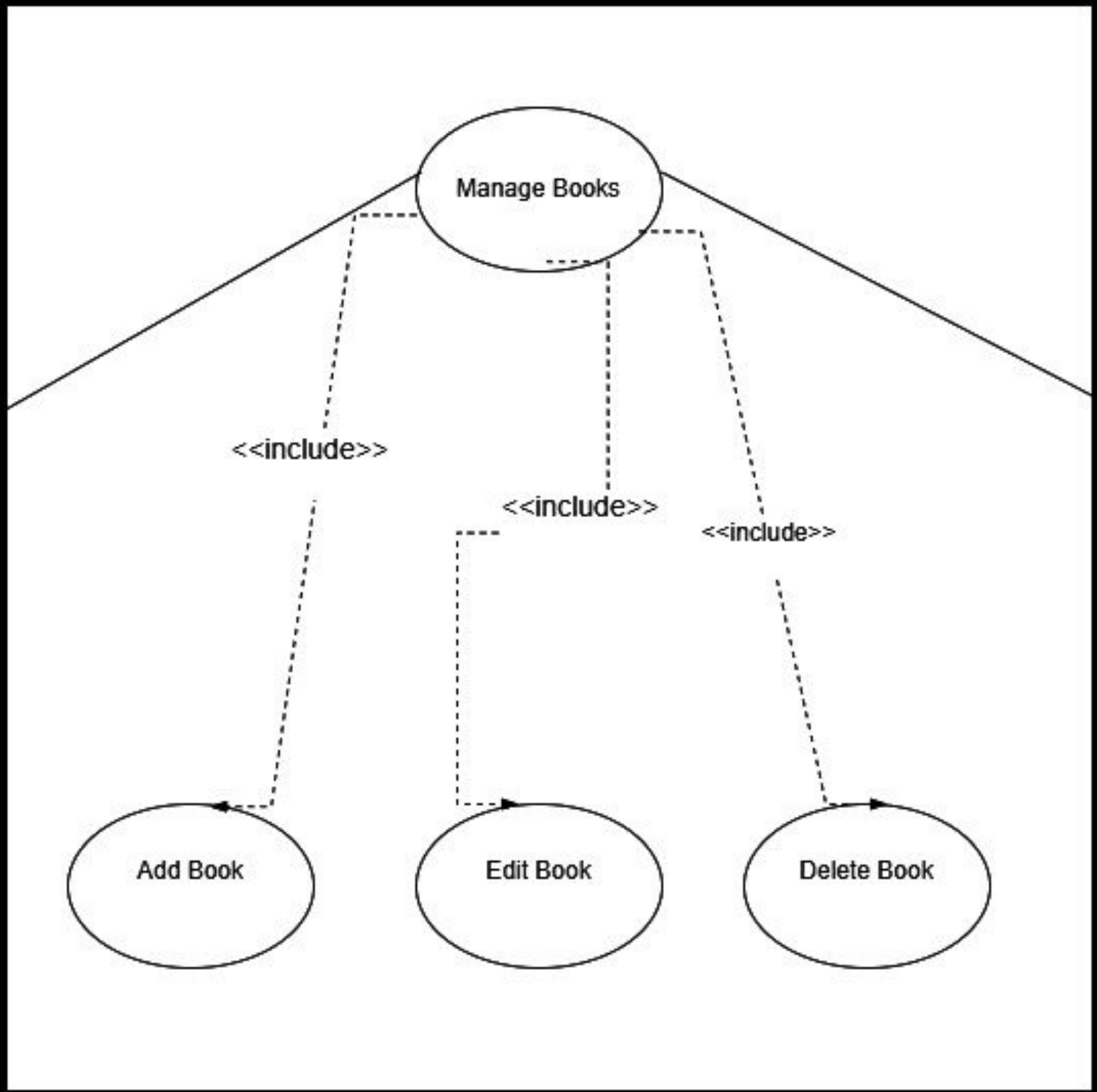
users(id, username, password, roles),
books(id, title, author, isbn, Copies),
borrows(id, user_id, book_id, bookTitle, userName, requestDate borrow_date, due_date, return_date, actaulReturnDate, requestStatus, borrowStatus)

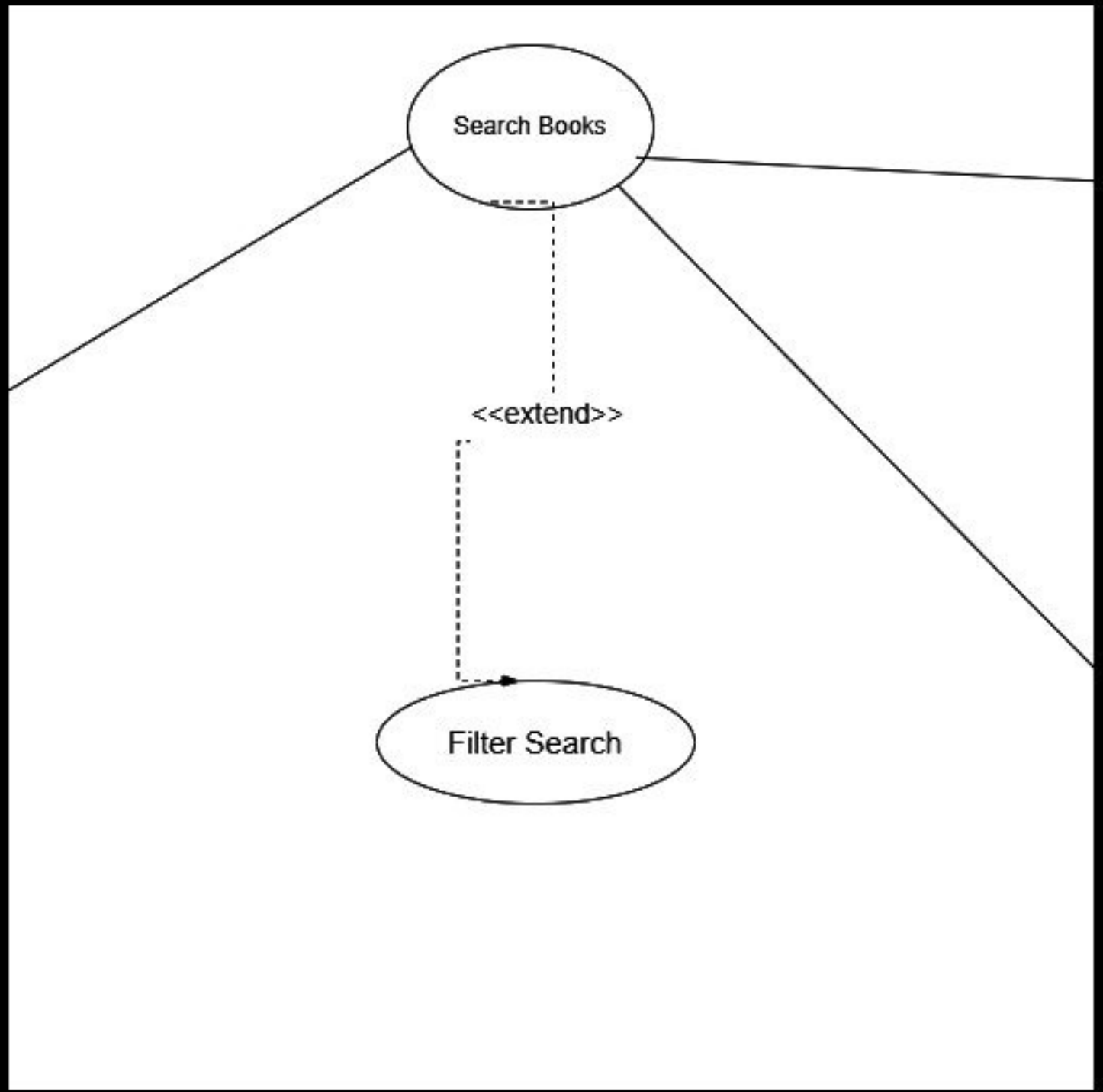
8.Security

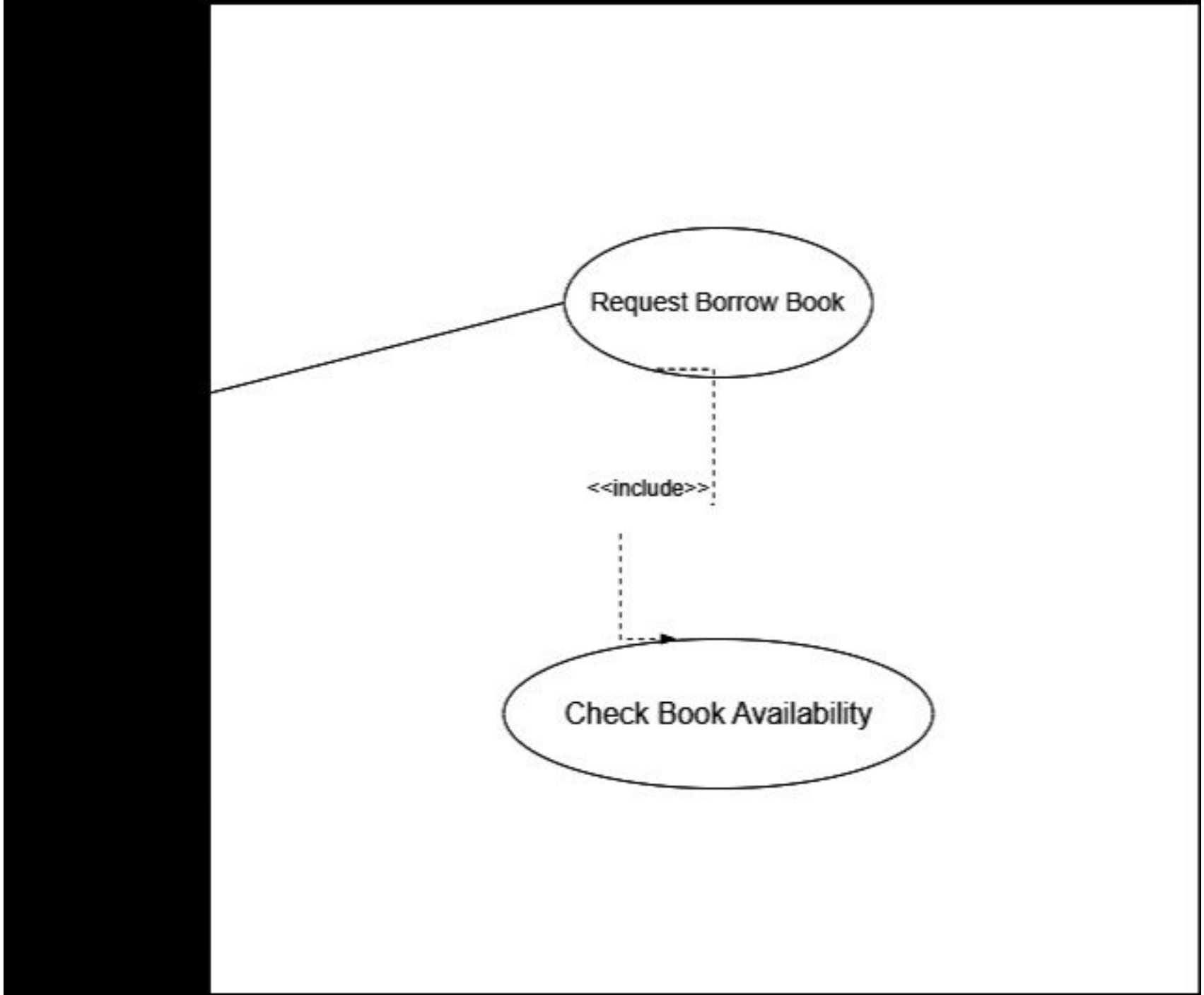
- JWT authentication
- Role-based endpoint authorization (@PreAuthorize)
- Passwords hashed (BCrypt)

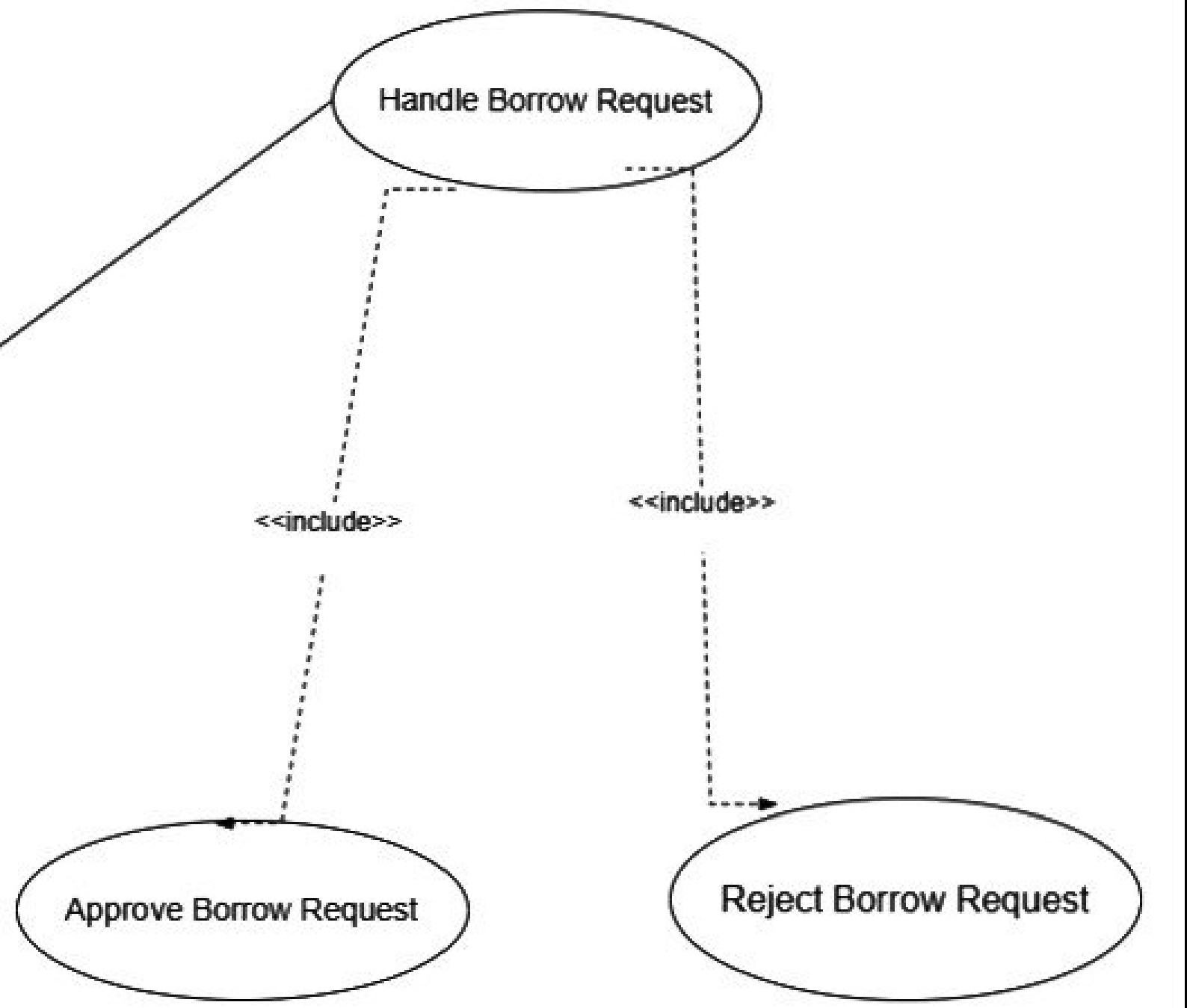

User Login
<<include>>
Validate Credentials
<<include>>
Generate JWT Token

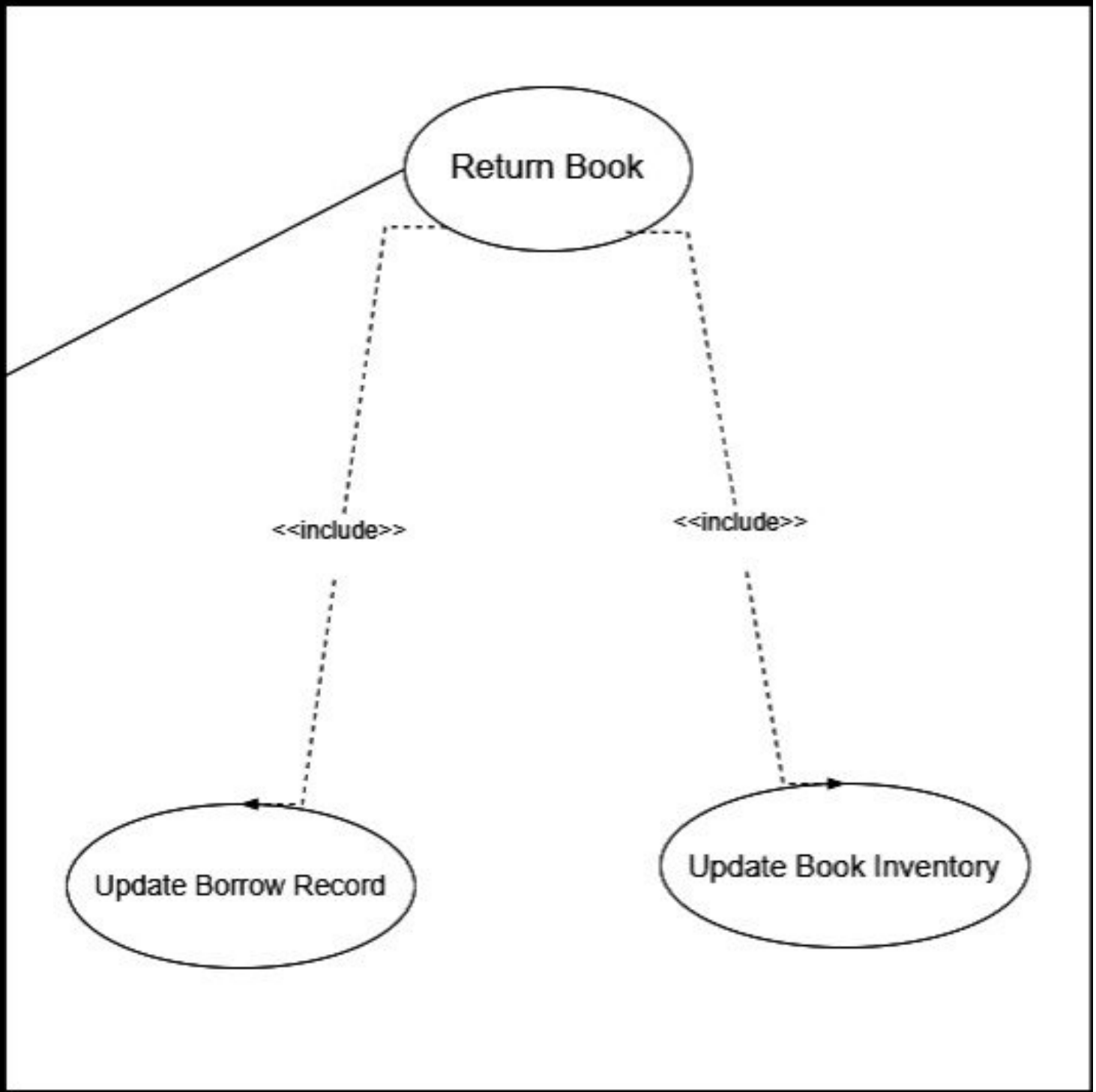

Manage Users
<<include>>
<<include>>
<<include>>
View User
Edit User
Delete User

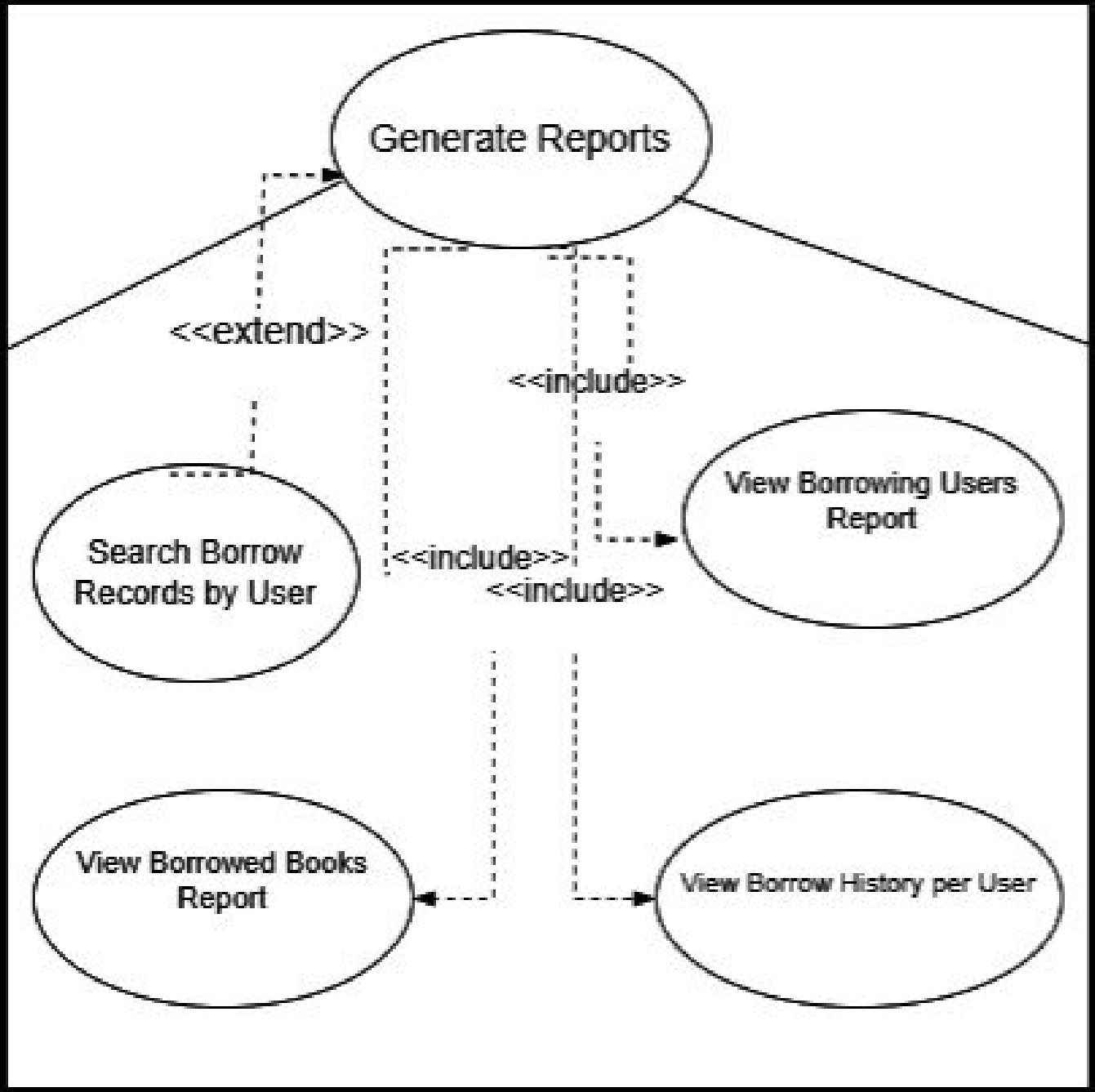

Manage Books
<<include>>
<<include>>
<<include>>
Add Book
Edit Book
Delete Book

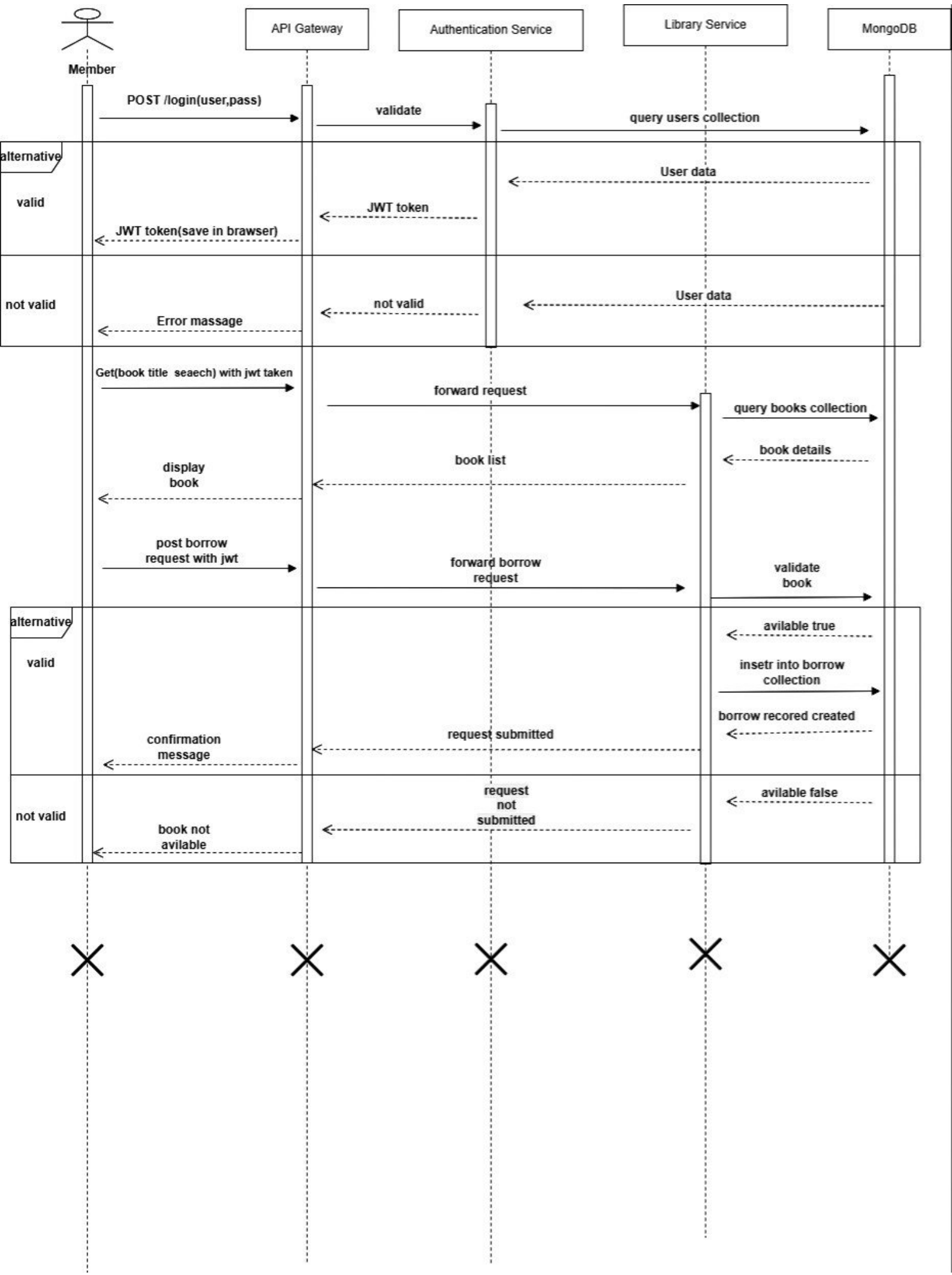

Search Books
<<extend>>
Filter Search

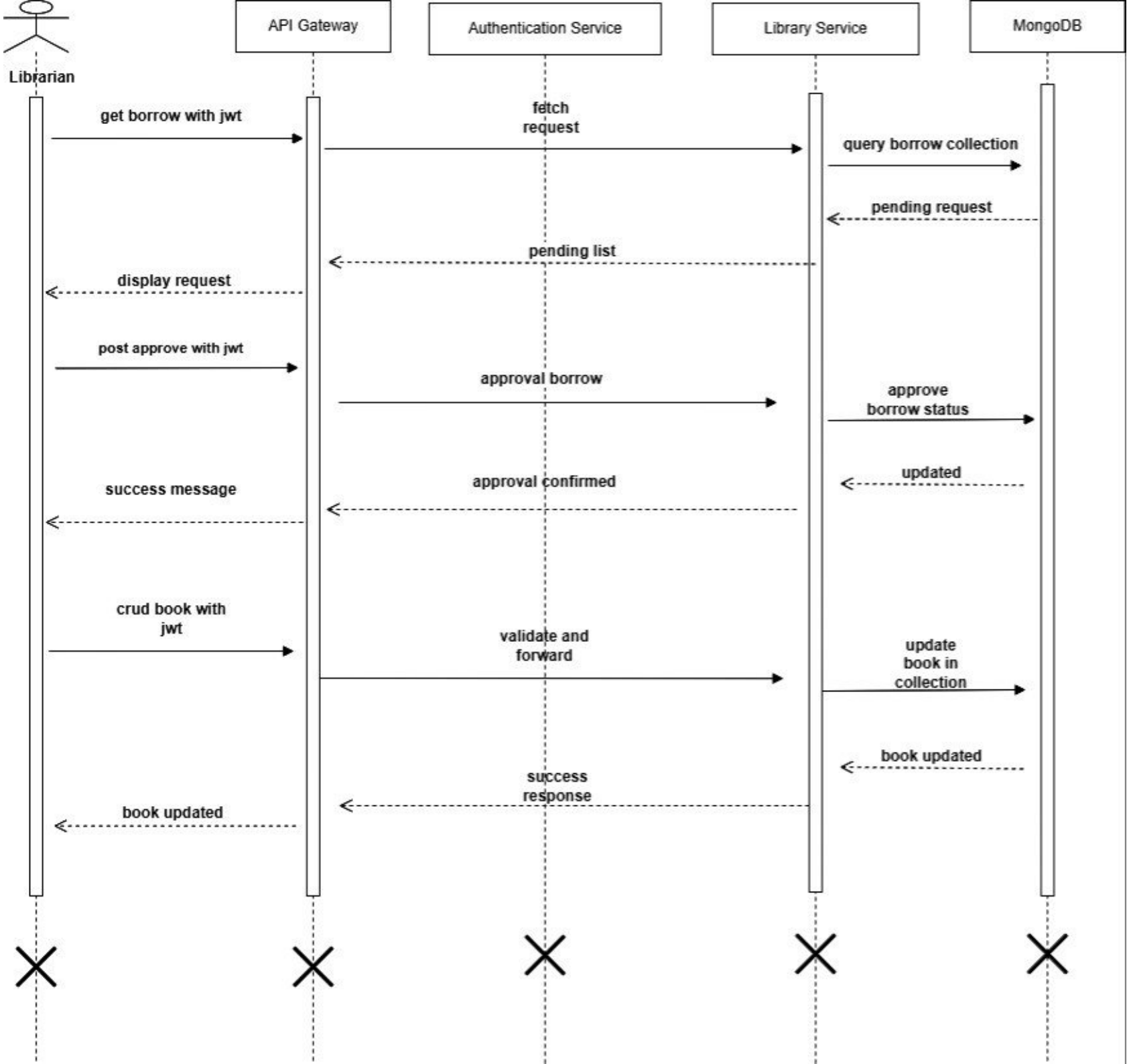

Request Borrow Book
<<include>>
Check Book Availability

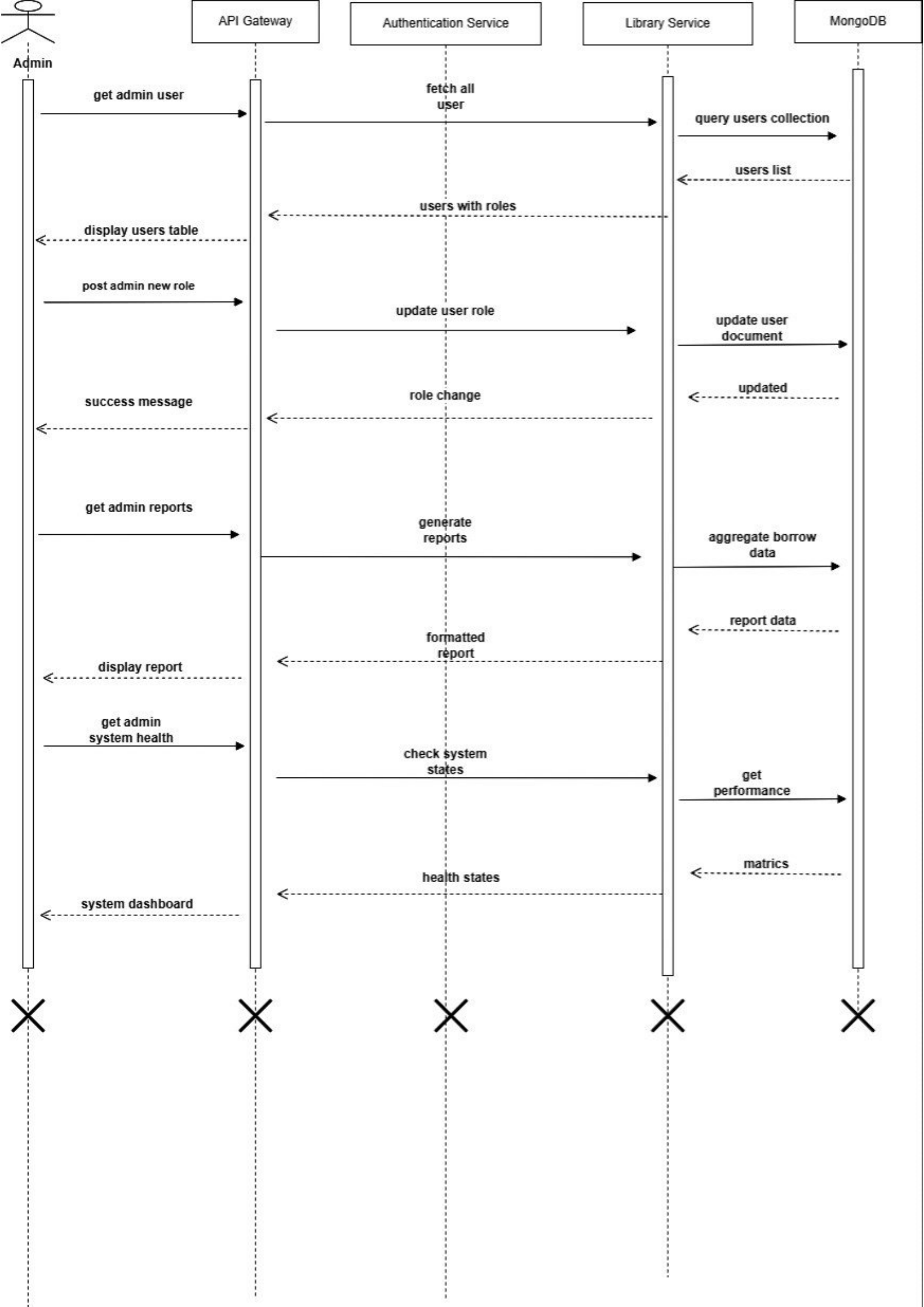

Handle Borrow Request
<<include>>
<<include>>
Approve Borrow Request
Reject Borrow Request

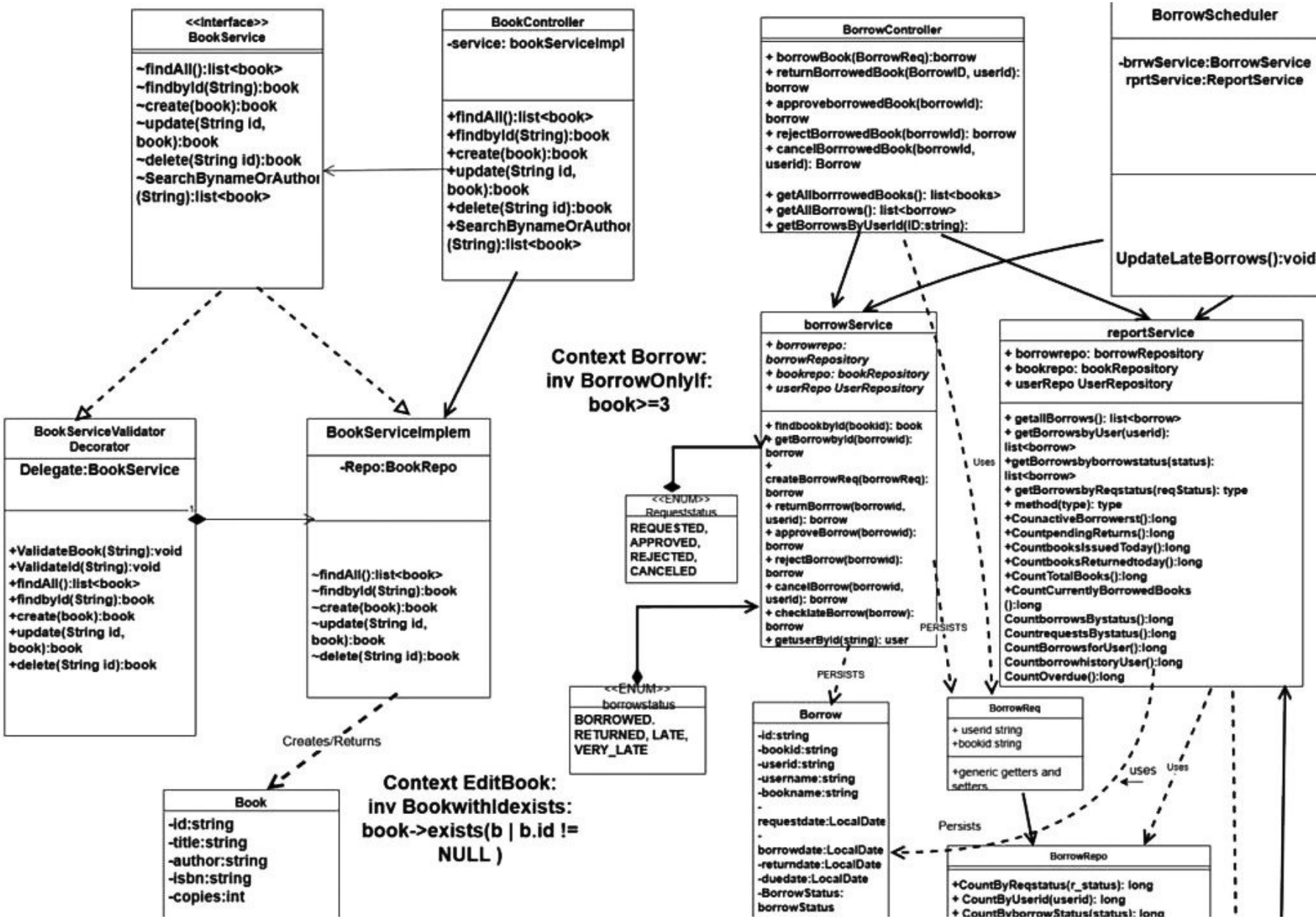

Return Book
<<include>>
<<include>>
Update Borrow Record
Update Book Inventory

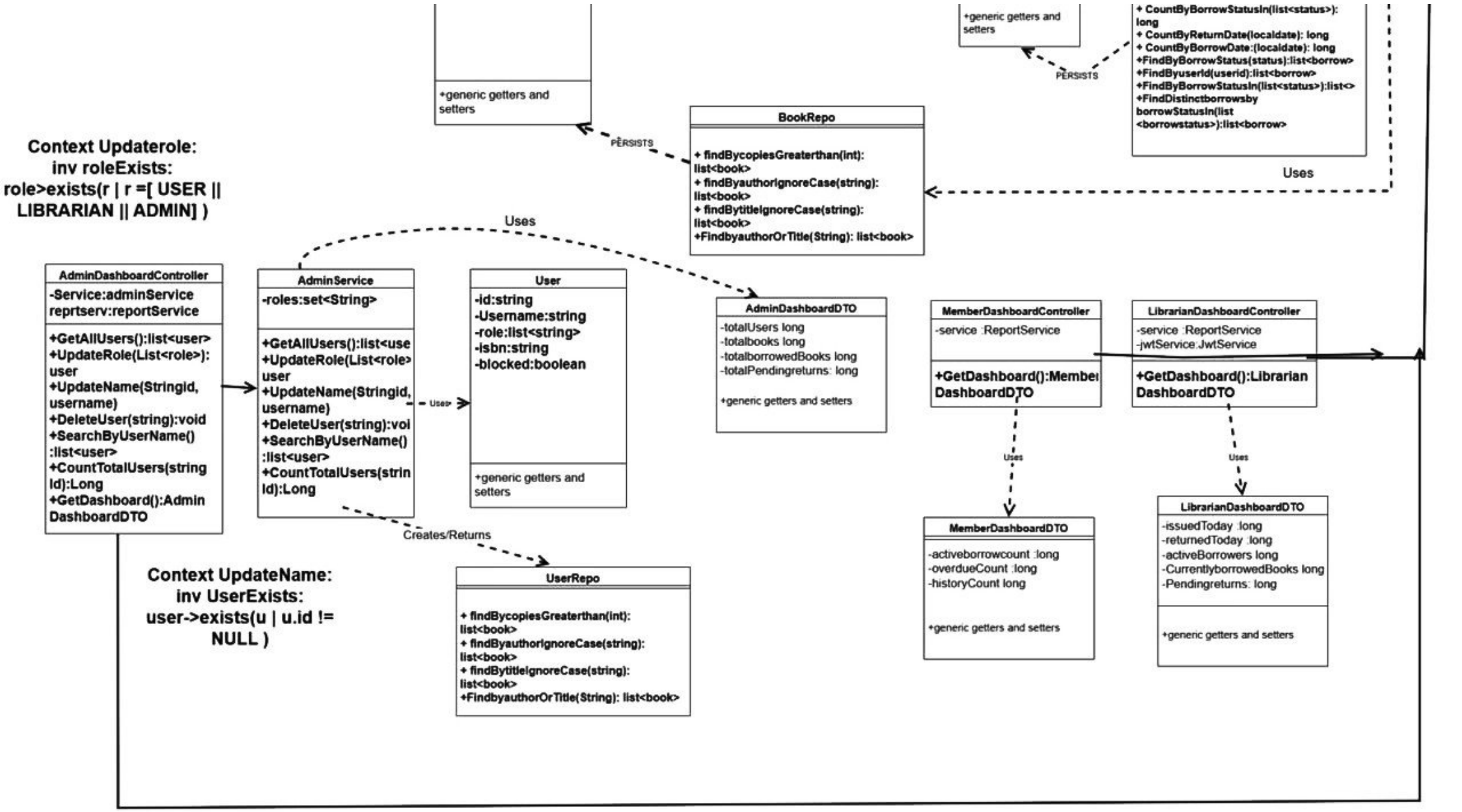

Generate Reports
<<extend>>
<<include>>
Search Borrow Records by User
<<include>>
<<include>>
View Borrowing Users Report
View Borrowed Books Report
View Borrow History per User

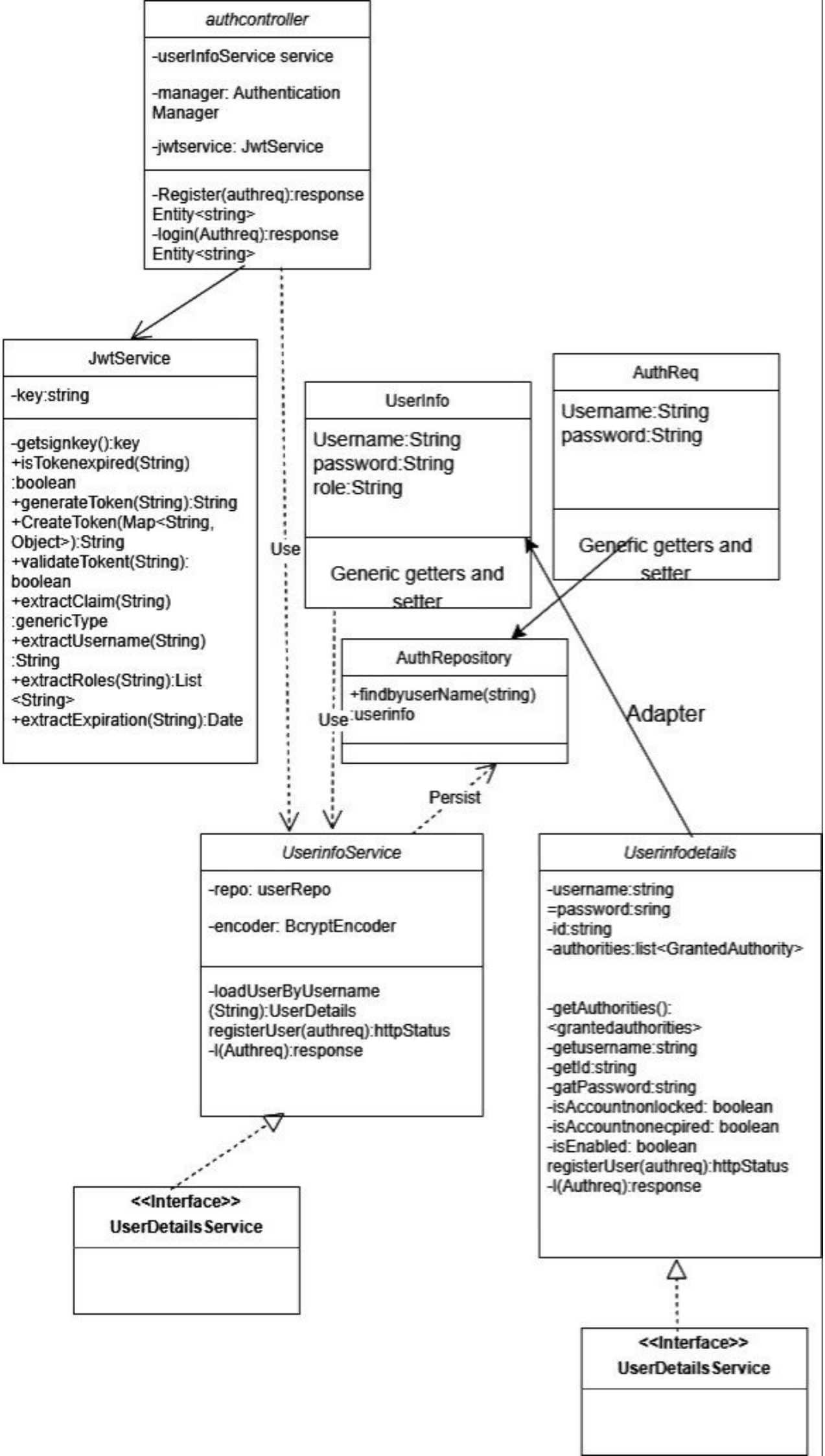

Member
API Gateway
Authentication Service
Library Service
MongoDB
POST /login(user,pass)
validate
query users collection
alternative
valid
User data
JWT token
JWT token(save in brawser)
not valid
not valid
User data
Error massage
Get(book title seaech) with jwt taken
forward request
query books collection
book details
book list
display
book
post borrow
request with jwt
forward borrow
request
validate
book
alternative
valid
avilable true
insetr into borrow
collection
borrow recored created
request submitted
confirmation
message
not valid
request
not
submitted
avilable false
book not
avilable

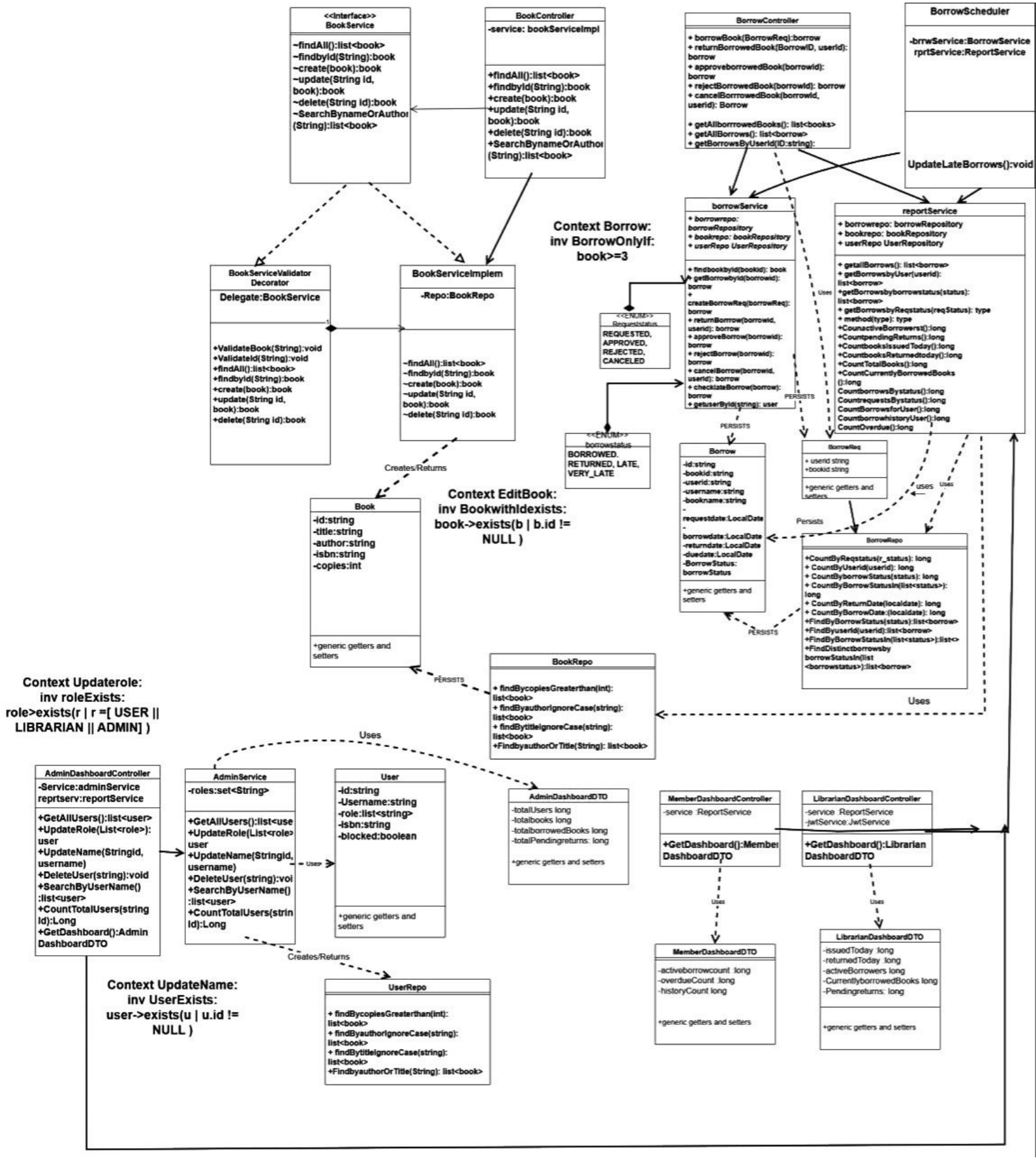

Librarian
API Gateway
Authentication Service
Library Service
MongoDB
get borrow with jwt
fetch request
query borrow collection
pending request
pending list
display request
post approve with jwt
approval borrow
approve borrow status
updated
approval confirmed
success message
crud book with jwt
validate and forward
update book in collection
book updated
success response
book updated


Admin
API Gateway
Authentication Service
Library Service
MongoDB
get admin user
fetch all user
query users collection
users list
users with roles
display users table
post admin new role
update user role
update user document
updated
role change
success message
get admin reports
generate reports
aggregate borrow data
report data
formatted report
display report
get admin system health
check system states
get performance
matrics
health states
system dashboard

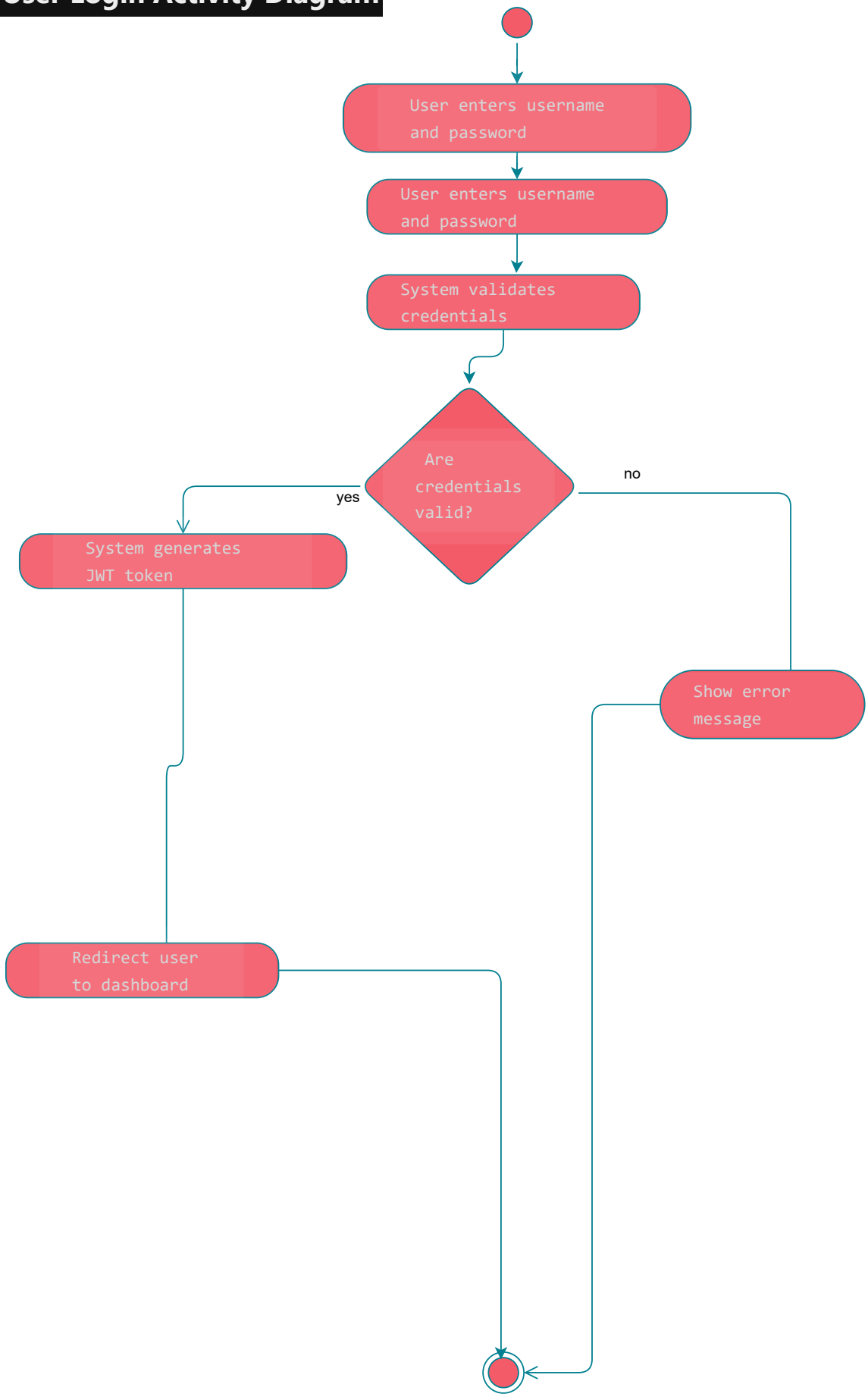

<<interface>>
BookService
~findAll():list<book>
~findbyId(String):book
~create(book):book
~update(String id, book):book
~delete(String id):book
~SearchBynameOrAuthor(String):list<book>
BookController
-service: bookServiceImpl
+findAll():list<book>
+findbyId(String):book
+create(book):book
+update(String id, book):book
+delete(String id):book
+SearchBynameOrAuthor(String):list<book>
BookServiceValidator Decorator
Delegate:BookService
+ValidateBook(String):void
+ValidateId(String):void
+findAll():list<book>
+findbyId(String):book
+create(book):book
+update(String id, book):book
+delete(String id):book
BookServiceImplem
-Repo:BookRepo
~findAll():list<book>
~findbyId(String):book
~create(book):book
~update(String id, book):book
~delete(String id):book
Creates/Returns
Book
-id:string
-title:string
-author:string
-isbn:string
-copies:int
Context Borrow:
inv BorrowOnlyIf:
book>=3
Context EditBook:
inv BookwithIdexists:
book->exists(b | b.id != NULL)
BorrowController
+ borrowBook(BorrowReq):borrow
+ returnBorrowedBook(BorrowID, userId): borrow
+ approveborrowedBook(borrowId): borrow
+ rejectBorrowedBook(borrowId): borrow
+ cancelBorrrowedBook(borrowId, userId): Borrow
+ getAllborrrowedBooks(): list<books>
+ getAllBorrows(): list<borrow>
+ getBorrowsByUserId(ID:string):
BorrowScheduler
-brrwService:BorrowService
rprtService:ReportService
UpdateLateBorrows():void
borrowService
+ borrowrepo: borrowRepository
+ bookrepo: bookRepository
+ userRepo UserRepository
+ findbookbyId(bookid): book
+ getBorrowbyId(borrowid): borrow
+ createBorrowReq(borrowReq): borrow
+ returnBorrrow(borrowid, userid): borrow
+ approveBorrow(borrowid): borrow
+ rejectBorrow(borrowid): borrow
+ cancelBorrow(borrowid, userid): borrow
+ checklateBorrow(borrow): borrow
+ getuserById(string): user
reportService
+ borrowrepo: borrowRepository
+ bookrepo: bookRepository
+ userRepo UserRepository
+ getallBorrows(): list<borrow>
+ getBorrowsbyUser(userid): list<borrow>
+getBorrowsbyborrowstatus(status): list<borrow>
+ getBorrowsbyReqstatus(reqStatus): type
+ method(type): type
+CounactiveBorrowerst():long
+CountpendingReturns():long
+CountbooksIssuedToday():long
+CountbooksReturnedtoday():long
+CountTotalBooks():long
+CountCurrentlyBorrowedBooks():long
CountborrowsBystatus():long
CountrequestsBystatus():long
CountBorrowsforUser():long
CountborrowhistoryUser():long
CountOverdue():long
<<ENUM>>
Requeststatus
REQUESTED,
APPROVED,
REJECTED,
CANCELED
<<ENUM>>
borrowstatus
BORROWED.
RETURNED, LATE,
VERY_LATE
Borrow
-id:string
-bookId:string
-userid:string
-username:string
-bookname:string
-requestdate:LocalDate
-borrowdate:LocalDate
-returndate:LocalDate
-duedate:LocalDate
-BorrowStatus: borrowStatus
BorrowReq
+ userid string
+bookid string
+generic getters and setters
BorrowRepo
+CountByReqstatus(r_status): long
+ CountByUserid(userid): long
Uses
PERSISTS
PERSISTS
uses
Uses
Persists


+generic getters and setters
PERSISTS
BookRepo
+ findBycopiesGreaterthan(int): list<book>
+ findByauthorIgnoreCase(string): list<book>
+ findBytitleIgnoreCase(string): list<book>
+FindbyauthorOrTitle(String): list<book>
+generic getters and setters
PERSISTS
+ CountByBorrowStatusIn(list<status>): long
+ CountByReturnDate(localdate): long
+ CountByBorrowDate:(localdate): long
+FindByBorrowStatus(status):list<borrow>
+FindByuserId(userid):list<borrow>
+FindByBorrowStatusIn(list<status>):list<>
+FindDistinctborrowsby borrowStatusIn(list <borrowstatus>):list<borrow>
Uses
Context Updaterole:
inv roleExists:
role>exists(r | r =[USER || LIBRARIAN || ADMIN])
Uses
AdminDashboardController
-Service:adminService
reprtserv:reportService
+GetAllUsers():list<user>
+UpdateRole(List<role>): user
+UpdateName(Stringid, username)
+DeleteUser(string):void
+SearchByUserName() :list<user>
+CountTotalUsers(string Id):Long
+GetDashboard():Admin DashboardDTO
AdminService
-roles:set<String>
+GetAllUsers():list<use
+UpdateRole(List<role> user
+UpdateName(Stringid, username)
+DeleteUser(string):voi
+SearchByUserName() :list<user>
+CountTotalUsers(strin Id):Long
User
User
-id:string
-Username:string
-role:list<string>
-isbn:string
-blocked:boolean
+generic getters and setters
AdminDashboardDTO
-totalUsers long
-totalbooks long
-totalborrowedBooks long
-totalPendingreturns: long
+generic getters and setters
MemberDashboardController
-service :ReportService
+GetDashboard():Member DashboardDTO
LibrarianDashboardController
-service :ReportService
-jwtService:JwtService
+GetDashboard():Librarian DashboardDTO
Uses
Uses
MemberDashboardDTO
-activeborrowcount :long
-overdueCount :long
-historyCount long
+generic getters and setters
LibrarianDashboardDTO
-issuedToday :long
-returnedToday :long
-activeBorrowers long
-CurrentlyborrowedBooks long
-Pendingreturns: long
+generic getters and setters
Creates/Returns
Context UpdateName:
inv UserExists:
user->exists(u | u.id != NULL)
UserRepo
+ findBycopiesGreaterthan(int): list<book>
+ findByauthorIgnoreCase(string): list<book>
+ findBytitleIgnoreCase(string): list<book>
+FindbyauthorOrTitle(String): list<book>

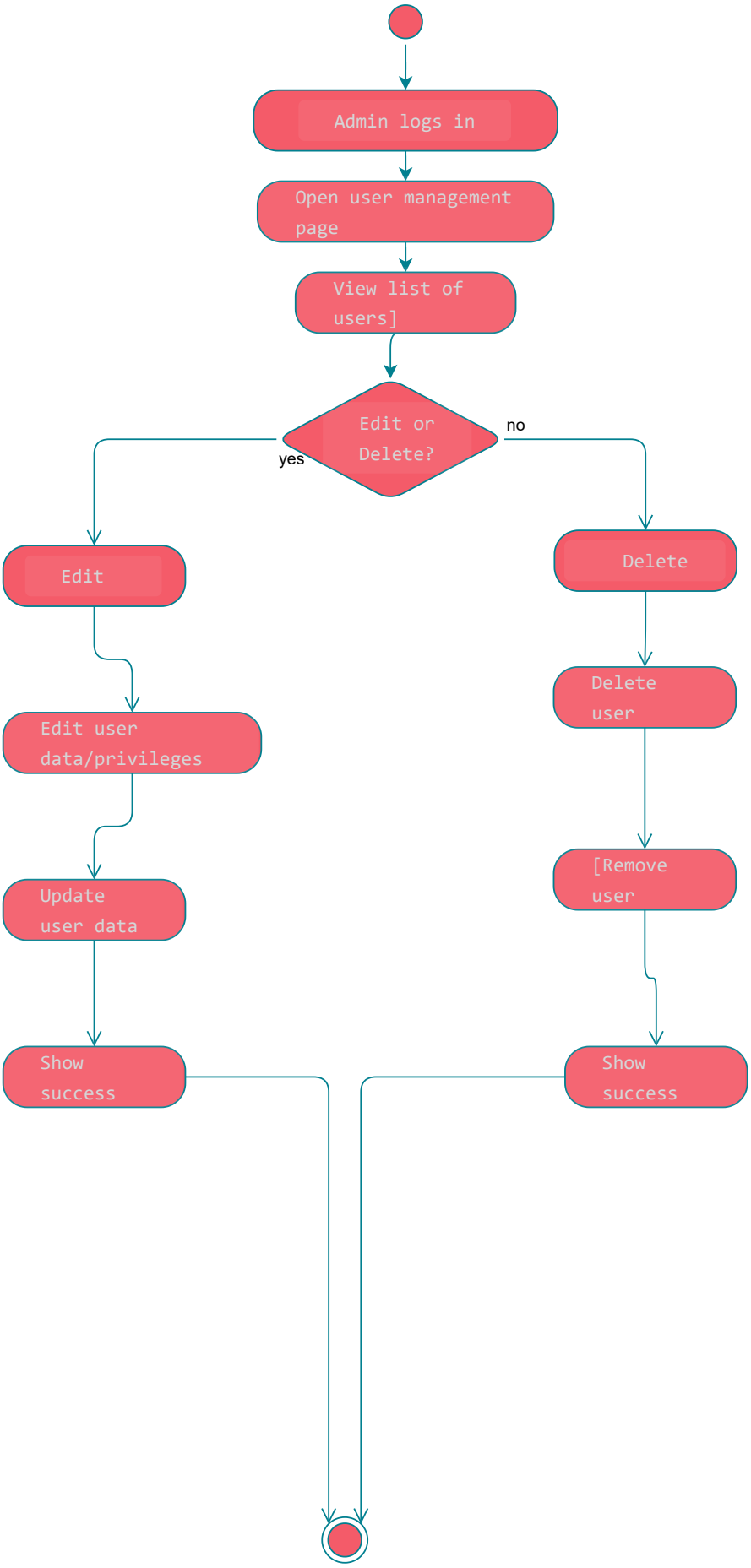

authcontroller
-userInfoService service
-manager: Authentication Manager
-jwtservice: JwtService
-Register(authreq):response Entity<string>
-login(Authreq):response Entity<string>
JwtService
-key:string
-getsignkey():key
+isTokenexpired(String) :boolean
+generateToken(String):String
+CreateToken(Map<String, Object>):String
+validateTokent(String): boolean
+extractClaim(String) :genericType
+extractUsername(String) :String
+extractRoles(String):List <String>
+extractExpiration(String):Date
Use
UserInfo
Username:String
password:String
role:String
Generic getters and setter
AuthReq
Username:String
password:String
Generic getters and setter
AuthRepository
+findbyuserName(string) :userinfo
Use
Adapter
Persist
UserinfoService
-repo: userRepo
-encoder: BcryptEncoder
-loadUserByUsername (String):UserDetails
registerUser(authreq):httpStatus
-l(Authreq):response
Userinfodetails
-username:string
=password:sring
-id:string
-authorities:list<GrantedAuthority>
-getAuthorities(): <grantedauthorities>
-getusername:string
-getId:string
-gatPassword:string
-isAccountnonlocked: boolean
-isAccountnonecpired: boolean
-isEnabled: boolean
registerUser(authreq):httpStatus
-l(Authreq):response
<<Interface>>
UserDetails Service
<<Interface>>
UserDetails Service


<<Interface>>
BookService
~findAll():list<book>
~findbyId(String):book
~create(book):book
~update(String id, book):book
~delete(String id):book
~SearchBynameOrAuthor(String):list<book>
BookController
-service: bookServiceImpl
+findAll():list<book>
+findbyId(String):book
+create(book):book
+update(String id, book):book
+delete(String id):book
+SearchBynameOrAuthor(String):list<book>
BorrowController
+ borrowBook(BorrowReq):borrow
+ returnBorrowedBook(BorrowID, userId): borrow
+ approveborrowedBook(borrowId): borrow
+ rejectBorrowedBook(borrowId): borrow
+ cancelBorrowedBook(borrowId, userId): Borrow
+ getAllborrrowedBooks(): list<books>
+ getAllBorrows(): list<borrow>
+ getBorrowsByUserId(ID:string):
BorrowScheduler
-brrwService:BorrowService
rprtService:ReportService
UpdateLateBorrows():void
BookServiceValidator Decorator
Delegate:BookService
+ValidateBook(String):void
+ValidateId(String):void
+findAll():list<book>
+findbyId(String):book
+create(book):book
+update(String id, book):book
+delete(String id):book
BookServiceImplem
-Repo:BookRepo
~findAll():list<book>
~findbyId(String):book
~create(book):book
~update(String id, book):book
~delete(String id):book
Creates/Returns
Context Borrow:
inv BorrowOnlyIf:
book>=3
borrowService
+ borrowrepo: borrowRepository
+ bookrepo: bookRepository
+ userRepo UserRepository
+ findbookbyId(bookid): book
+ getBorrowbyId(borrowid): borrow
+ createBorrowReq(borrowReq): borrow
+ returnBorrow(borrowId, userid): borrow
+ approveBorrow(borrowid): borrow
+ rejectBorrow(borrowid): borrow
+ cancelBorrow(borrowId, userid): borrow
+ checklateBorrow(borrow): borrow
+ getuserByid(string): user
reportService
+ borrowrepo: borrowRepository
+ bookrepo: bookRepository
+ userRepo UserRepository
+ getallBorrows(): list<borrow>
+ getBorrowsbyUser(userId): list<borrow>
+getBorrowsbyborrowstatus(status): list<borrow>
+ getBorrowsbyReqstatus(reqStatus): type
+ method(type): type
+CounactiveBorrowers():long
+CountpendingReturns():long
+CountbooksIssuedToday():long
+CountbooksReturnedtoday():long
+CountTotalBooks():long
+CountCurrentlyBorrowedBooks():long
CountborrowsBystatus():long
CountrequeststsBystatus():long
CountBorrowsforUser():long
CountborrowhistoryUser():long
CountOverdue():long
<<ENUM>>
RequestStatus
REQUESTED, APPROVED, REJECTED, CANCELED
<<ENUM>>
borrowstatus
BORROWED. RETURNED, LATE, VERY_LATE
Borrow
-id:string
-bookId:string
-userid:string
-username:string
-bookname:string
-requestdate:LocalDate
-borrowdate:LocalDate
-returndate:LocalDate
-duedate:LocalDate
-BorrowStatus: borrowStatus
+generic getters and setters
BorrowReq
+ userId string
+bookid string
+generic getters and setters
Persists
PERSISTS
Uses
Context EditBook:
inv BookwithIdexists:
book->exists(b | b.id != NULL)
Book
-id:string
-title:string
-author:string
-isbn:string
-copies:int
+generic getters and setters
BorrowRepo
+CountByReqstatus(r_status): long
+ CountByUserid(userid): long
+ CountByborrowStatus(status): long
+ CountByBorrowStatusIn(list<status>): long
+ CountByReturnDate(localdate): long
+ CountByBorrowDate:(localdate): long
+FindByBorrowStatus(status):list<borrow>
+FindByuserId(userid):list<borrow>
+FindByBorrowStatusIn(list<status>):list<>
+FindDistinctborrowsby borrowStatusIn(list <borrowstatus>):list<borrow>
BookRepo
+ findBycopiesGreaterthan(int): list<book>
+ findByauthorIgnoreCase(string): list<book>
+ findBytitleIgnoreCase(string): list<book>
+FindbyauthorOrTitle(String): list<book>
Context Updaterole:
inv roleExists:
role>exists(r | r =[USER || LIBRARIAN || ADMIN])
AdminDashboardController
-Service:adminService
reprtserv:reportService
+GetAllUsers():list<user>
+UpdateRole(List<role>): user
+UpdateName(StringId, username)
+DeleteUser(string):void
+SearchByUserName() :list<user>
+CountTotalUsers(string Id):Long
+GetDashboard():Admin DashboardDTO
AdminService
-roles:set<String>
+GetAllUsers():list<use
+UpdateRole(List<role> user
+UpdateName(StringId, username)
+DeleteUser(string):voi
+SearchByUserName() :list<user>
+CountTotalUsers(strin Id):Long
User
-id:string
-Username:string
-role:list<string>
-isbn:string
-blocked:boolean
+generic getters and setters
AdminDashboardDTO
-totalUsers long
-totalbooks long
-totalborrowedBooks long
-totalPendingreturns: long
+generic getters and setters
MemberDashboardController
-service :ReportService
+GetDashboard():Member DashboardDTO
LibrarianDashboardController
-service :ReportService
-jwtService:JwtService
+GetDashboard():Librarian DashboardDTO
MemberDashboardDTO
-activeborrowcount :long
-overdueCount :long
-historyCount long
+generic getters and setters
LibrarianDashboardDTO
-issuedToday :long
-returnedToday :long
-activeBorrowers long
-CurrentlyborrowedBooks long
-Pendingreturns: long
+generic getters and setters
Creates/Returns
Context UpdateName:
inv UserExists:
user->exists(u | u.id != NULL)
UserRepo
+ findBycopiesGreaterthan(int): list<book>
+ findByauthorIgnoreCase(string): list<book>
+ findBytitleIgnoreCase(string): list<book>
+FindbyauthorOrTitle(String): list<book>

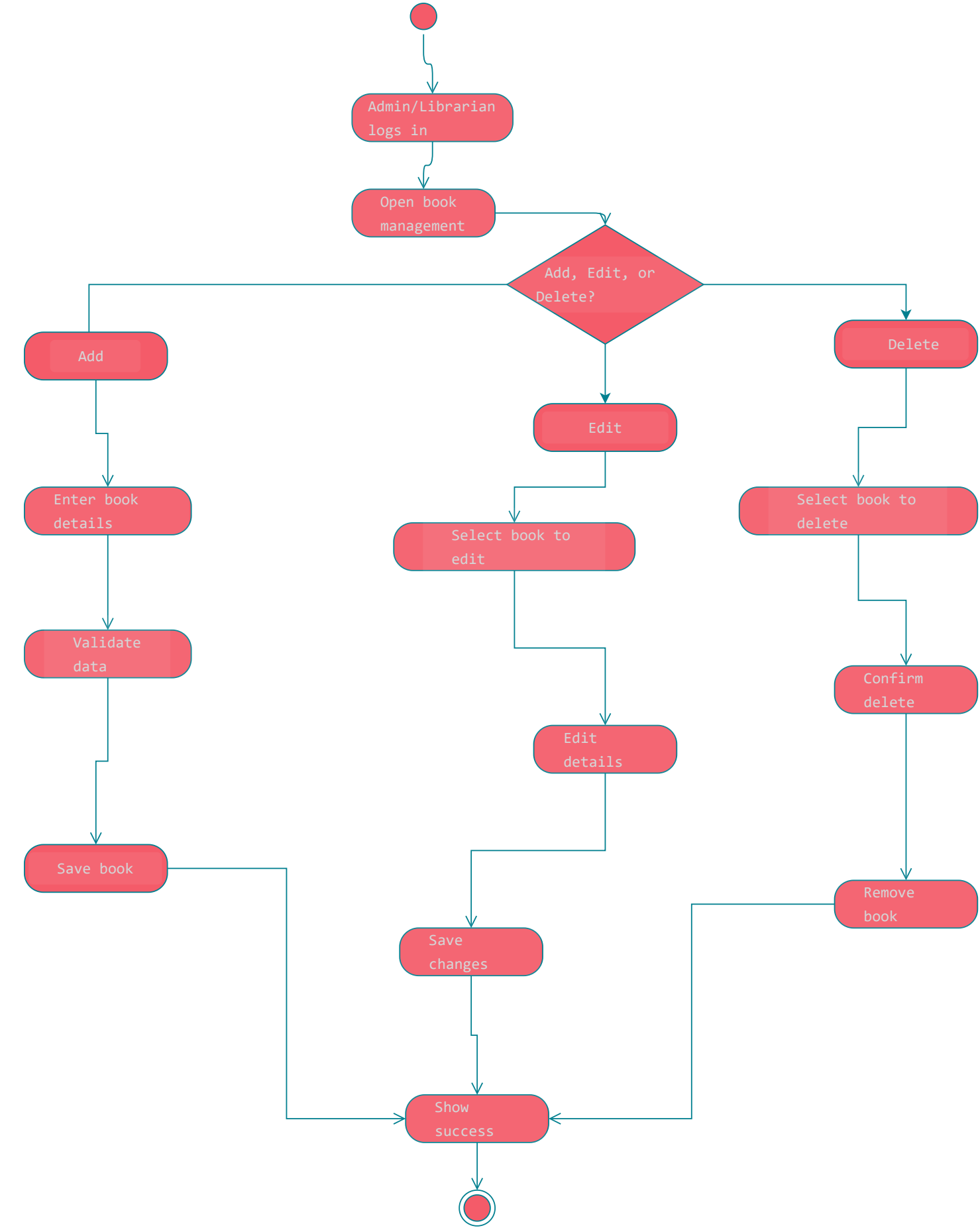
1. User Login Activity Diagram



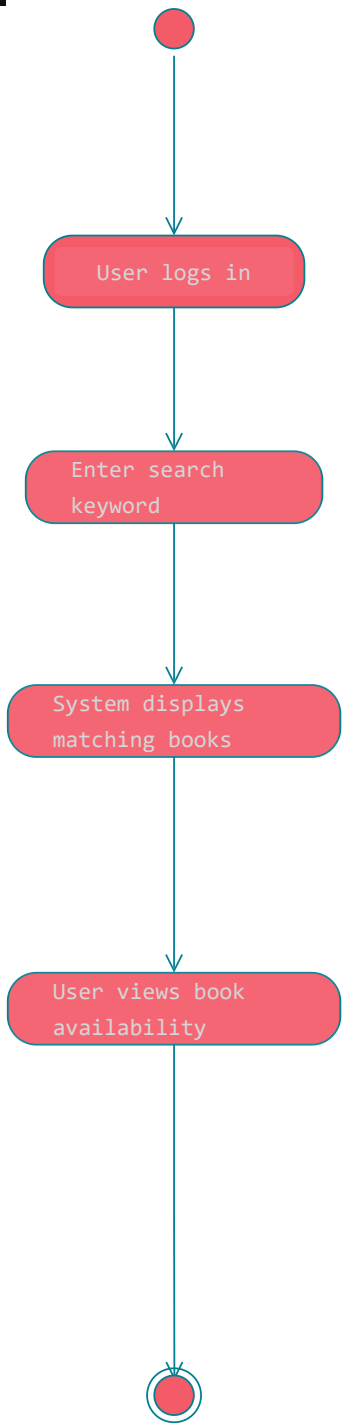
2. Manage Users Activity Diagram (Admin)



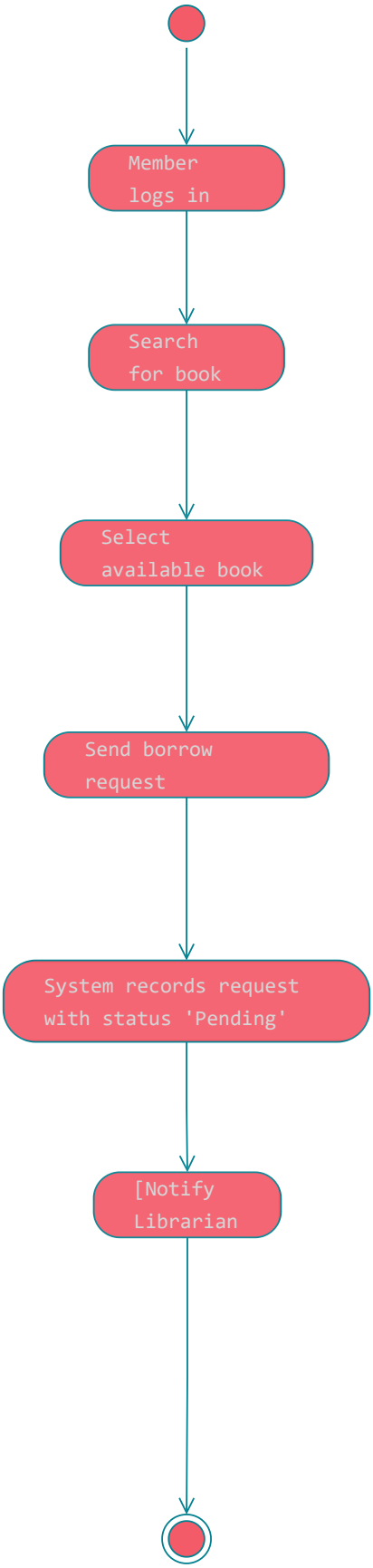
3. Manage Books Activity Diagram (Admin/Librarian)



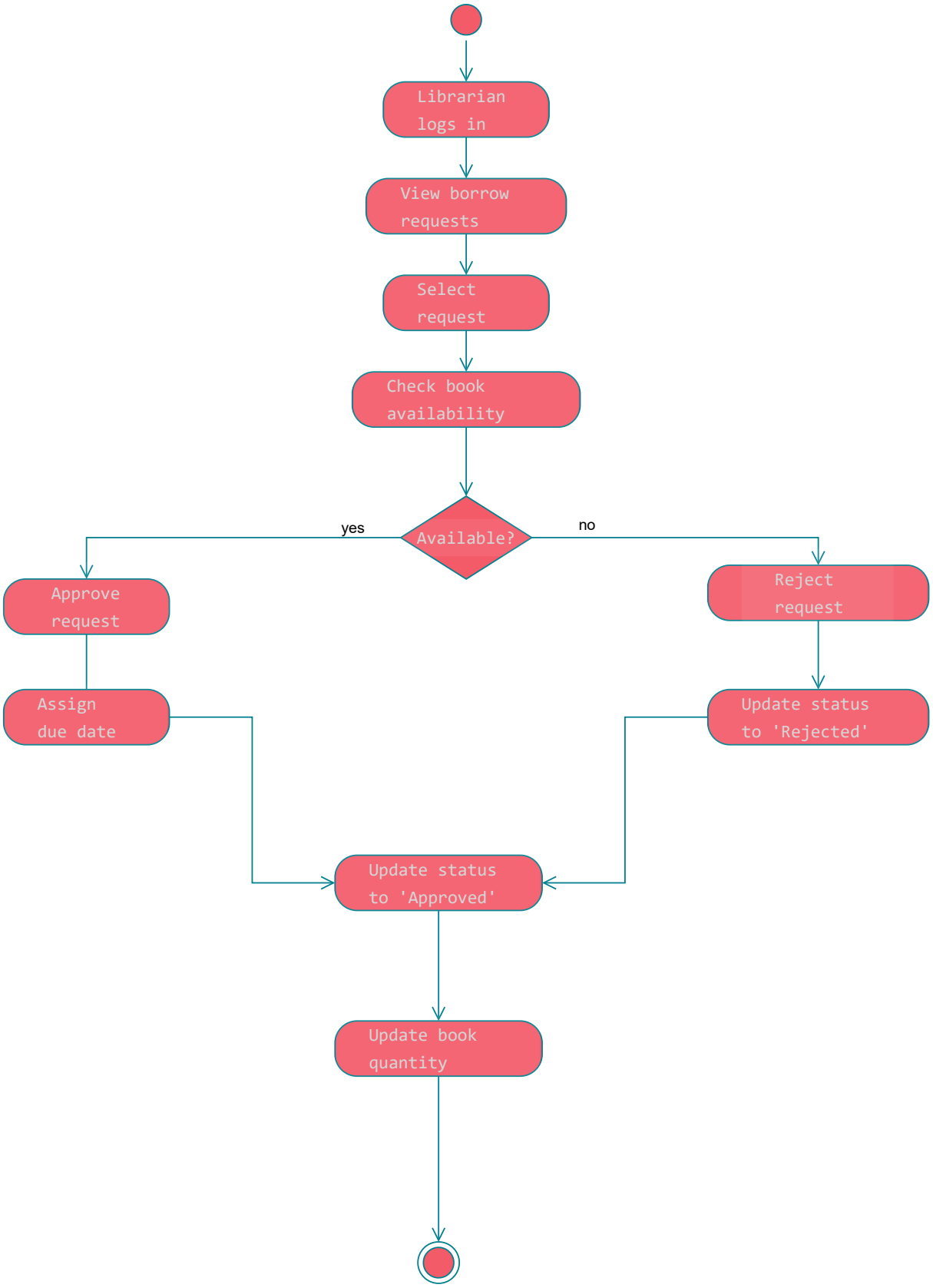
4. Search Books Activity Diagram



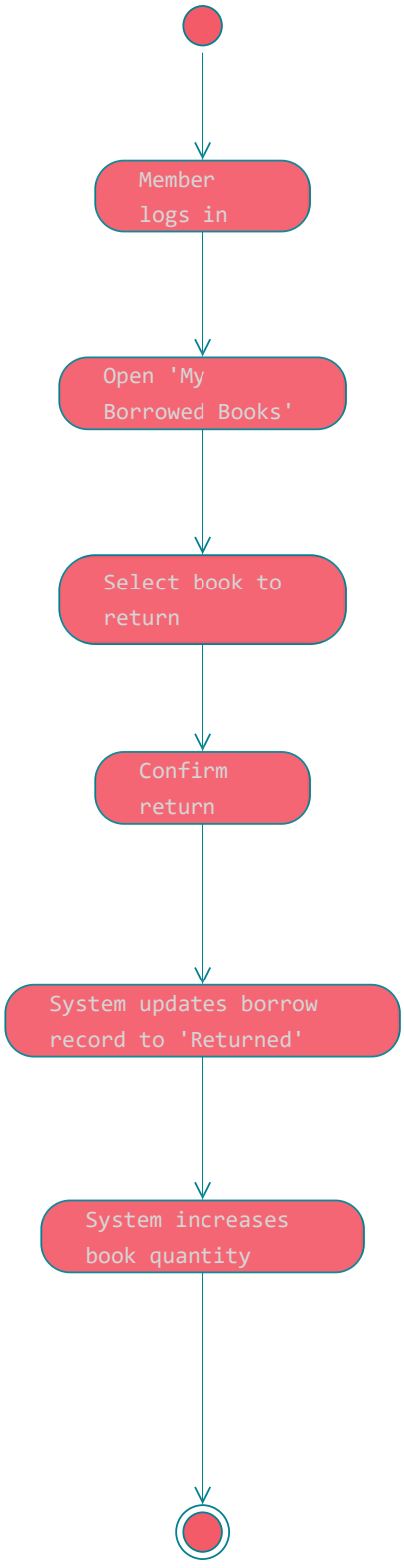
5. Request Borrow Book Activity Diagram (Member



6. Approve/Reject Borrow Request Activity Diagram (Librarian)



7. Return Book Activity Diagram (Member)



8. Generate Reports Activity Diagram (Admin/Librarian)