SOFTWARE ENGINEERING 2

Library Project

ABOUT OUR PROJECT

- A simple library project in Java Spring Boot is a great way to get started with creating a web application for managing books and library patrons. This project will include features for adding and managing books and patrons, as well as loaning books to patrons. The application will use Spring Boot as the framework, along with Spring Data JPA for data persistence, and a simple REST API for client interactions.

PROJECT STRUCTURE

- Entities:
- Book: Represents a book in the library. It should have attributes such as id (unique identifier), title, author, isbn, and available (indicating if the book is available for loan).
- Patron: Represents a patron who can borrow books. It should have attributes such as id (unique identifier), name, and email.
- Loan: Represents the association between a book and a patron when a book is loaned. It should have attributes such as id, book (the book being loaned), patron (the patron who borrowed the book), loanDate (the date when the loan started), and returnDate (the date when the loan is expected to end).
- Repositories:
- BookRepository: Handles CRUD operations for the Book entity. It uses Spring Data JPA to interact with the database.
- PatronRepository: Handles CRUD operations for the Patron entity using Spring Data JPA.
- LoanRepository: Handles CRUD operations for the Loan entity using Spring Data JPA.
-

- Services:
- BookService: Provides business logic for managing books. This includes methods for adding new books, updating book details, deleting books, and finding books by various criteria.
- PatronService: Provides business logic for managing patrons. This includes methods for adding new patrons, updating patron details, deleting patrons, and finding patrons by various criteria.
- LoanService: Provides business logic for managing loans. This includes methods for loaning books to patrons, returning books, and finding active loans.

- Controllers:
- BookController: Exposes REST API endpoints for managing books. This includes endpoints for getting a list of books, getting a book by its ID, adding a new book, updating book details, and deleting a book.
- PatronController: Exposes REST API endpoints for managing patrons. This includes endpoints for getting a list of patrons, getting a patron by their ID, adding a new patron, updating patron details, and deleting a patron.
- LoanController: Exposes REST API endpoints for managing loans. This includes endpoints for loaning a book to a patron, returning a book, and getting a list of active loans.

- Database:
- Use an embedded database such as H2 for simplicity, or you can use a more production-ready database like PostgreSQL or MySQL.
- Configure Spring Boot to use JPA for database interactions.
- Security:
- Implement basic authentication for accessing the REST API endpoints to ensure only authorized users can perform certain actions.
- Use Spring Security to handle user authentication and authorization.

- Testing:
- Write unit tests for the services and repositories.
- Write integration tests for the controllers.
- Application:
- Create a main class with a main method that runs the Spring Boot application.
- Configure application properties such as the server port and database connection.

FUNCTIONAL REQUIREMENTS

- Add Book: The system should allow authorized users to add new books to the library's collection. Required details include the title, author, and ISBN.
- Update Book: The system should allow authorized users to update details of existing books, such as title, author, and availability status.
- Delete Book: The system should allow authorized users to delete books from the library's collection.
- List Books: The system should allow users to retrieve a list of all books in the library, with options for pagination.
- Search Books: The system should allow users to search for books by title, author, or ISBN

- Loan Management:
- Loan Book: The system should allow authorized users to loan books to patrons. This includes specifying the book and patron details, as well as the loan date.
- Return Book: The system should allow authorized users to record the return of a book, updating the loan status and availability of the book.
- List Loans: The system should allow users to retrieve a list of all active loans in the library, with options for pagination.

- User Authentication: The system should authenticate users before allowing access to certain actions such as adding, updating, or deleting books or patrons.
- User Authorization: The system should authorize users based on roles or permissions for performing specific actions

- Authentication and Authorization:

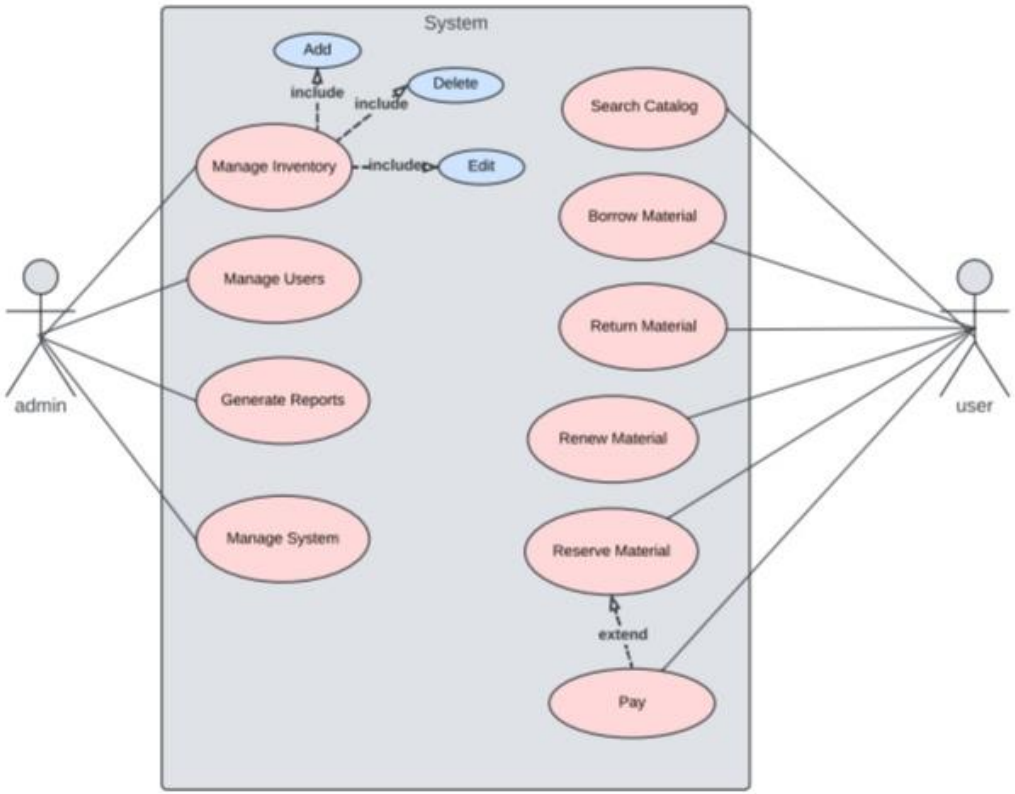
NON-FUNCTIONAL REQUIREMENTS

- Performance:
- The system should handle multiple concurrent requests efficiently.
- Response times for REST API endpoints should be within acceptable limits (e.g., <1 second).
- Scalability:
- The system should be scalable to handle an increasing number of users, books, patrons, and loans.
- Security:
- The system should ensure data privacy and protection, particularly for patron data.
- The system should protect against common security vulnerabilities such as SQL injection and cross-site scripting (XSS).

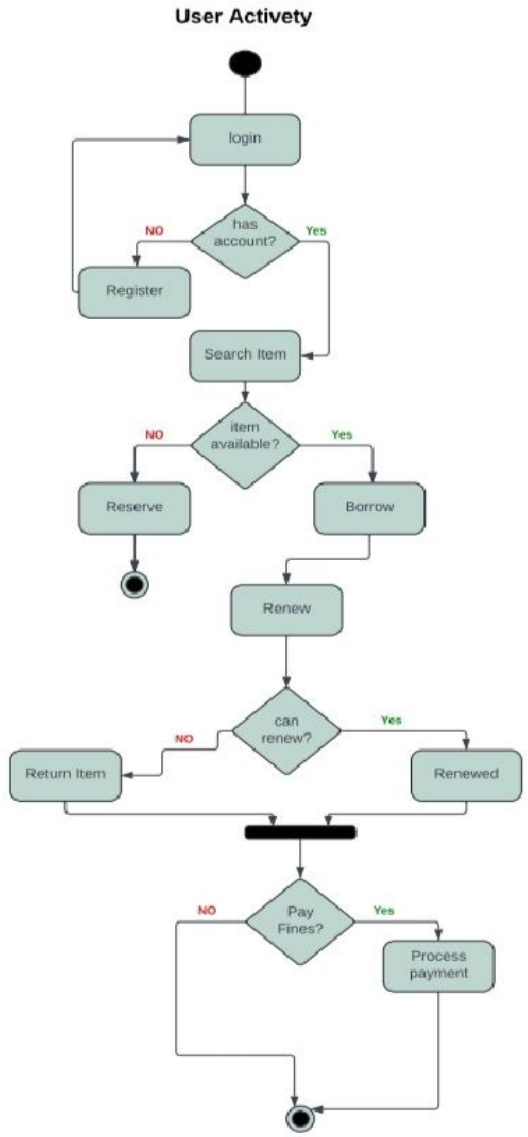
- Usability:
- REST API endpoints should follow a consistent naming convention and adhere to RESTful principles.
- Error responses should be informative and provide clear explanations.
- Maintainability:
- Code should be modular, following clean coding practices.
- The application should be easy to configure and customize.
- Reliability:
- The system should be reliable, with minimal downtime.
- Regular backups of the database should be made.

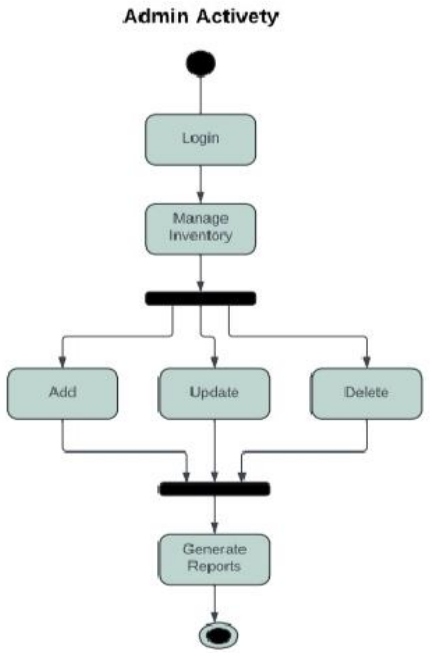
UML DIAGRAMS

- USECASE DIAGRAM

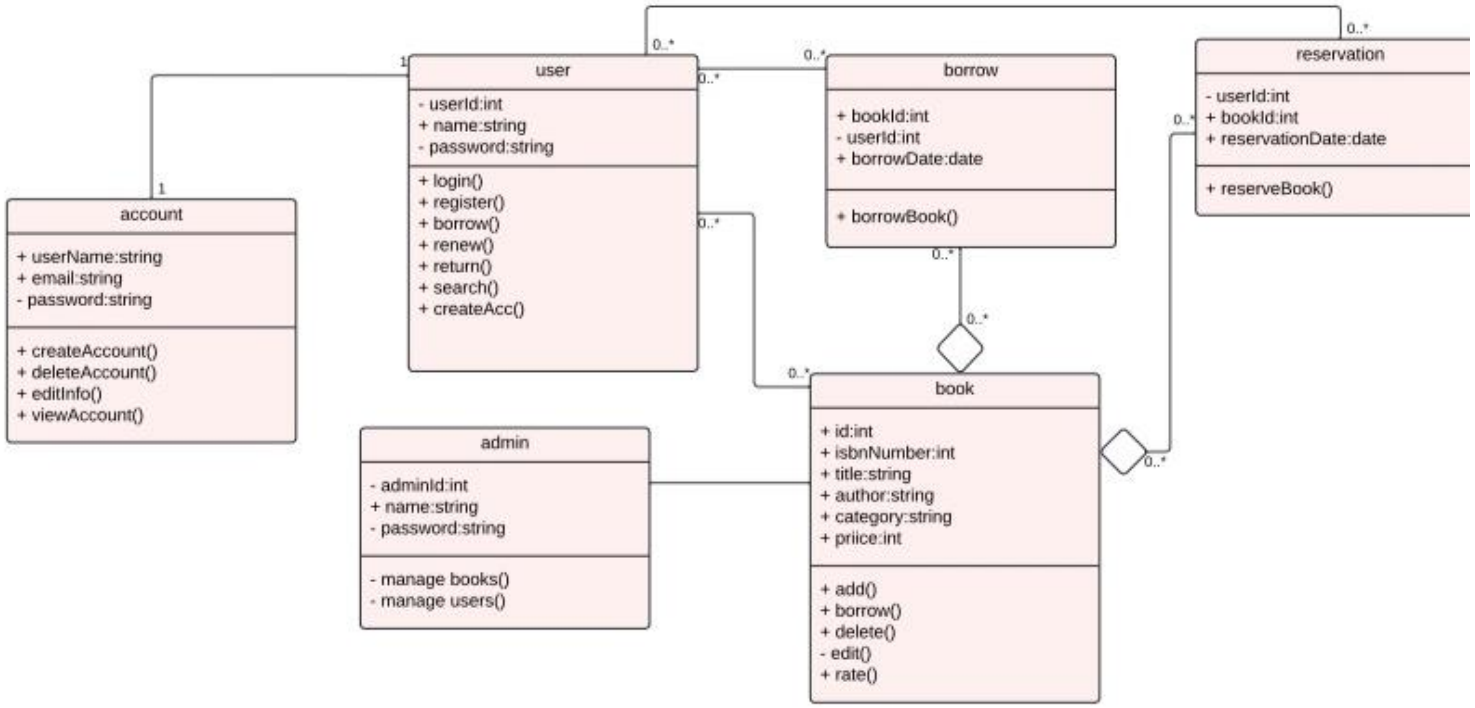


- ACTIVITY DIAGRAM

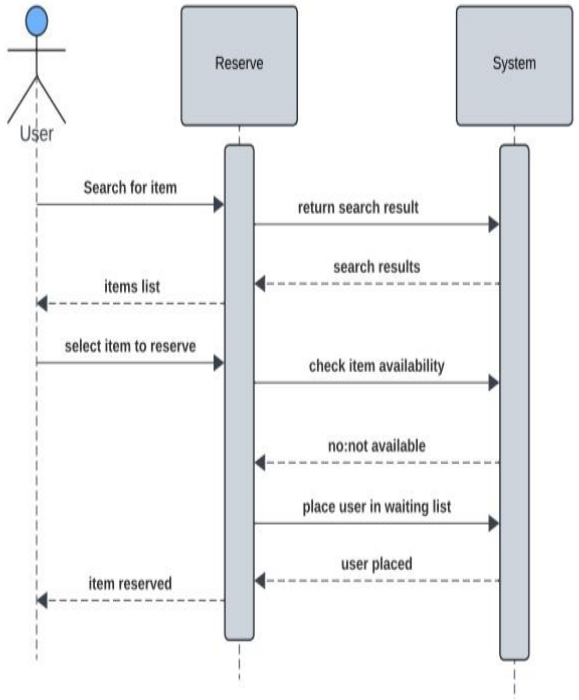


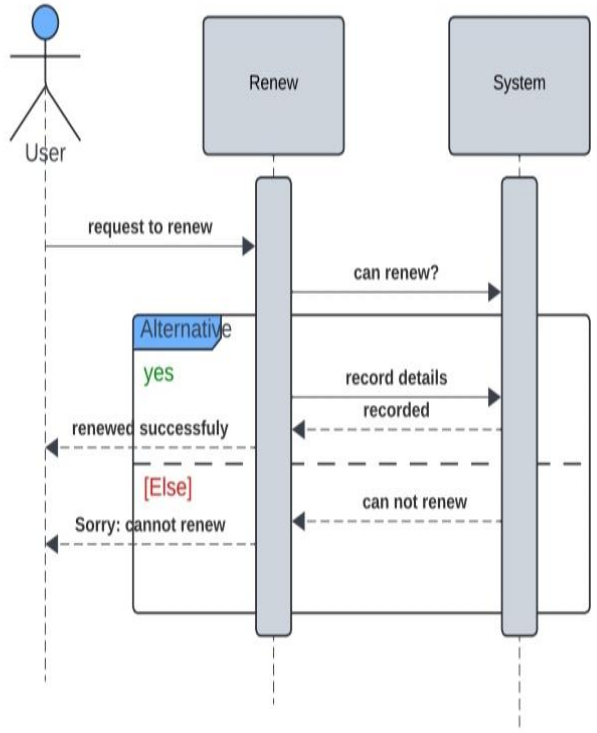


- CLASS DIAGRAM

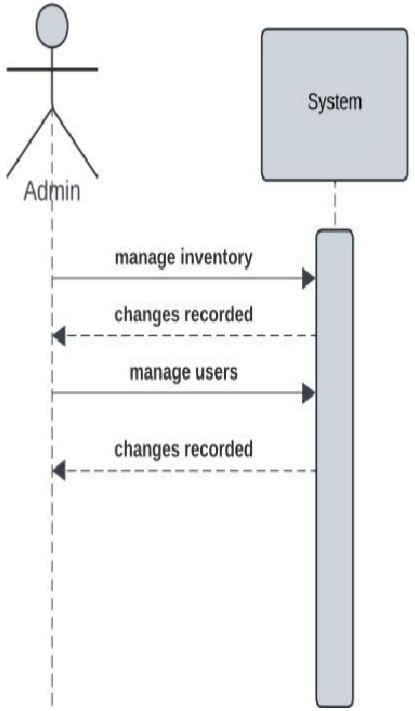


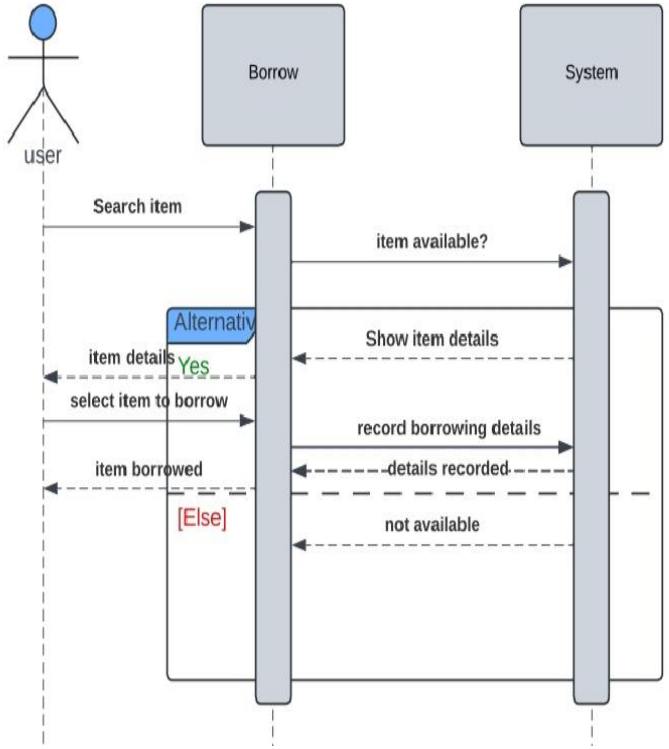
- SEQUENCE DIAGRAM



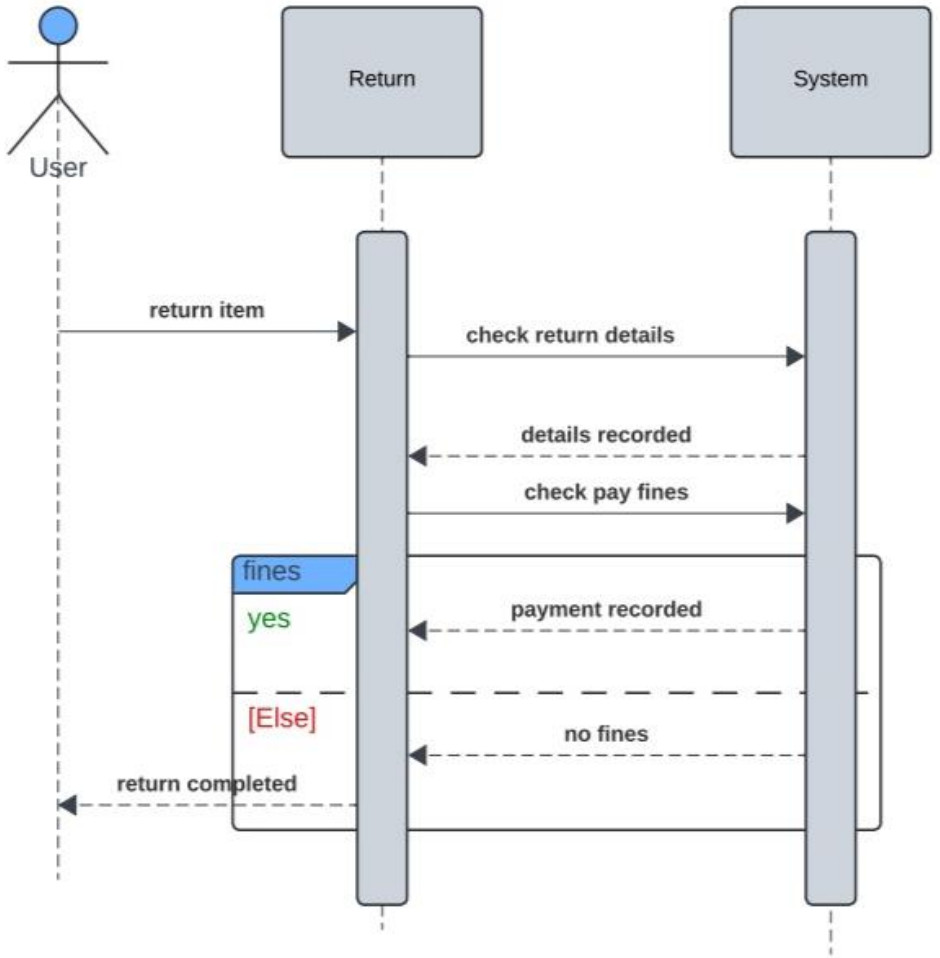


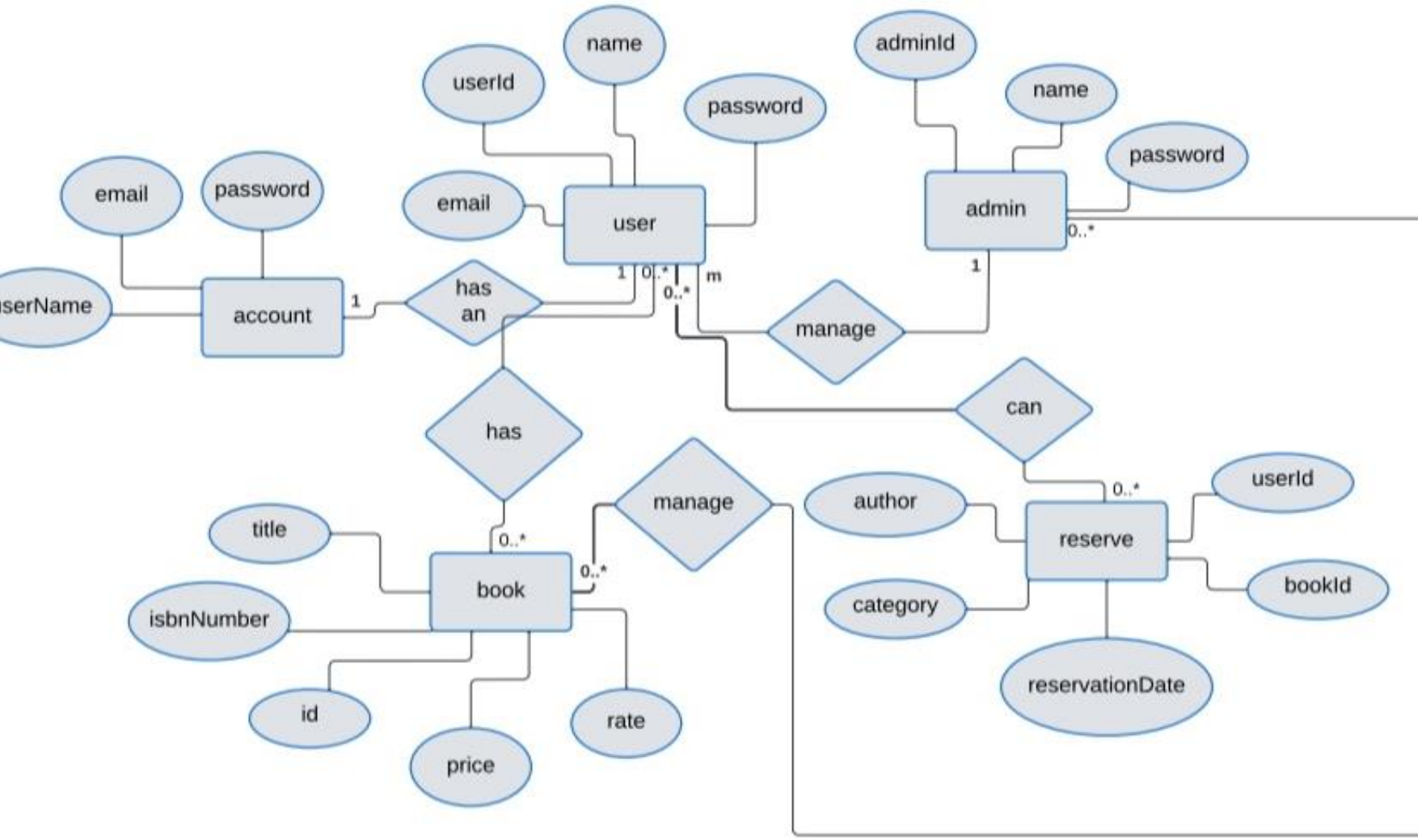
- SEQUENCE DIAGRAM





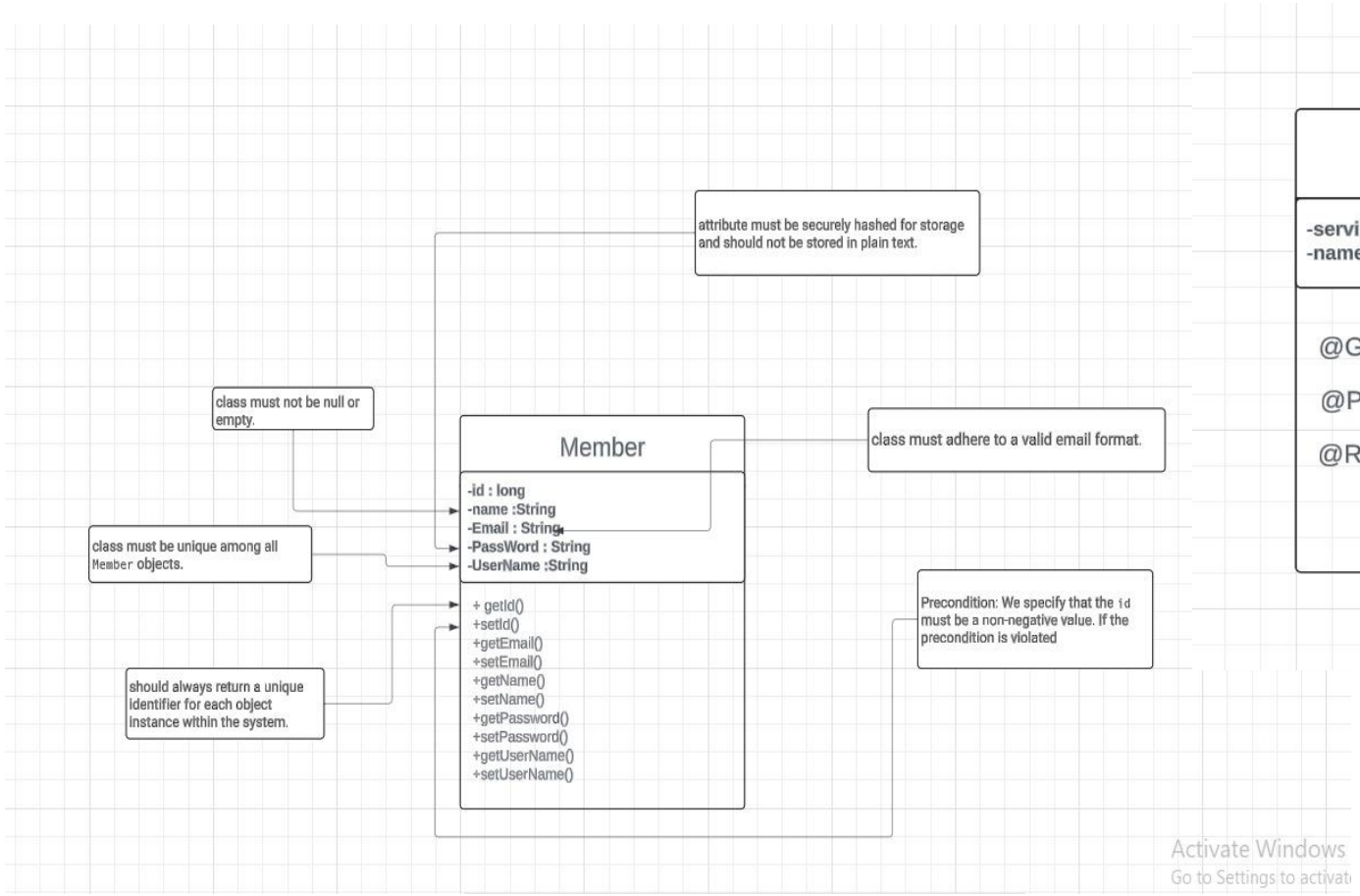
- SEQUENCE DIAGRAM

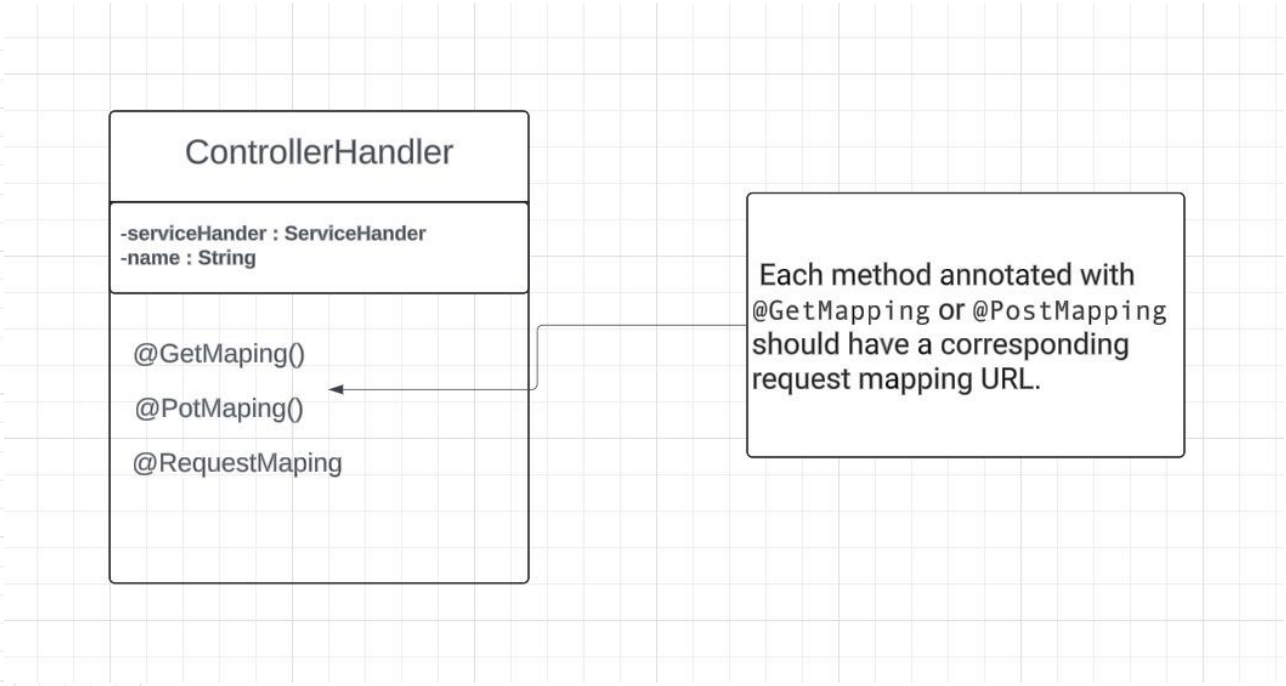



name
userId
password
adminId
name
email
password
password
email
user
admin
serName
account
1
has
an
1
0..*
0..*
m
1
0..*
manage
has
can
0..*
manage
author
userId
title
0..*
reserve
0..*
book
bookId
isbnNumber
category
id
price
rate
reservationDate

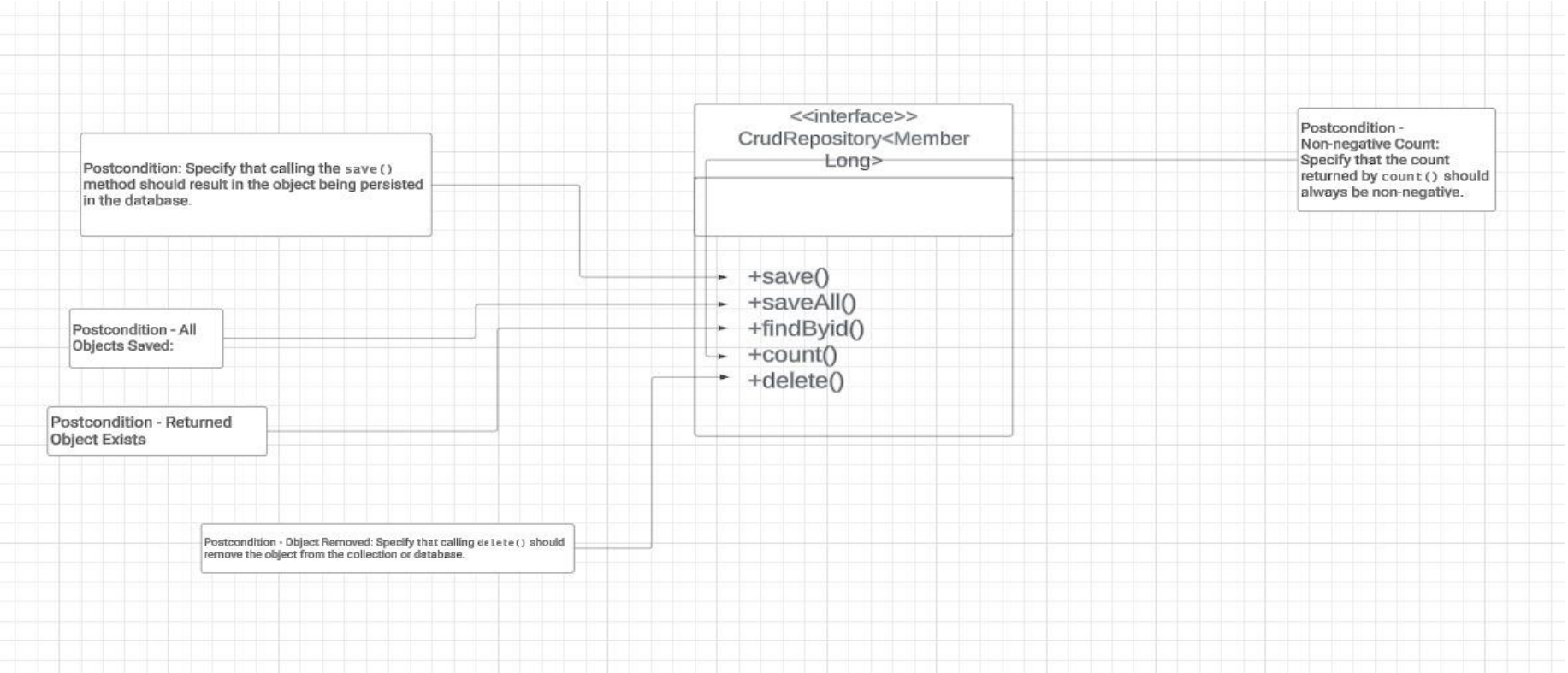
- ERD DIAGRAM

OCL

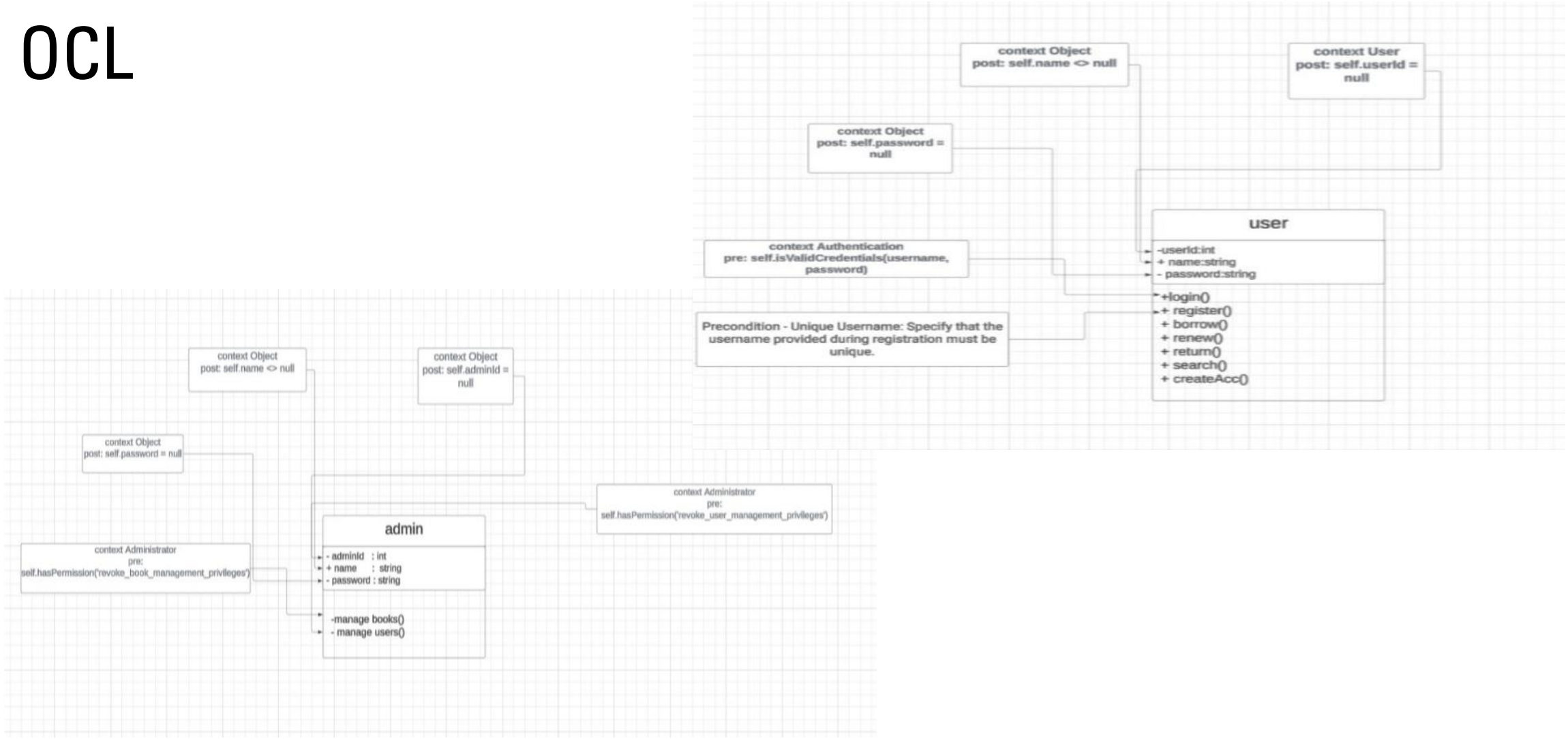




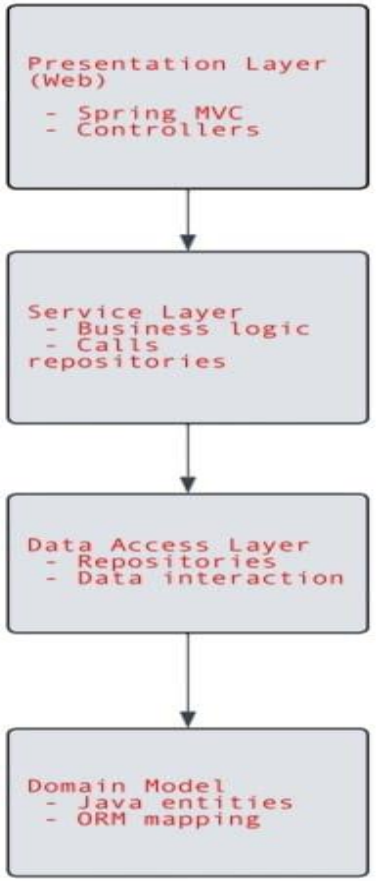
OCL


<<interface>>
CrudRepository<Member
Long>
+save()
+saveAll()
+findByid()
+count()
+delete()
Postcondition: Specify that calling the save() method should result in the object being persisted in the database.
Postcondition - All Objects Saved:
Postcondition - Returned Object Exists
Postcondition - Object Removed: Specify that calling delete() should remove the object from the collection or database.
Postcondition - Non-negative Count: Specify that the count returned by count() should always be non-negative.

OCL


context Object
post: self.name <> null
context User
post: self.userId = null
context Object
post: self.password = null
user
-userId:int
+ name:string
- password:string
context Authentication
pre: self.isValidCredentials(username, password)
+login()
+ register()
+ borrow()
+ renew()
+ return()
+ search()
+ createAcc()
Precondition - Unique Username: Specify that the username provided during registration must be unique.
context Object
post: self.name <> null
context Object
post: self.adminId = null
context Object
post: self.password = null
context Administrator
pre:
self.hasPermission('revoke_user_management_privileges')
admin
- adminId : int
+ name : string
- password : string
context Administrator
pre:
self.hasPermission('revoke_book_management_privileges')
-manage books()
- manage users()

ARCHITECTURE



- 1. Presentation Layer:
- Components: Spring MVC Controllers
- Purpose: Handles HTTP requests, maps them to appropriate service methods, and returns responses to the client.
- Connections: Communicates with the Service Layer.
- 2. Service Layer:
- Components: Service classes (e.g., LibraryService)
- Purpose: Contains business logic and coordinates operations between the presentation and data access layers.
- Connections: Interacts with both the Presentation Layer (for business processes) and the Data Access Layer (to fetch and manipulate data).

- 3. Data Access Layer:
- Components: Spring Data JPA or JDBC repositories
- Purpose: Manages interactions with the database and performs CRUD operations.
- Connections: Communicates with the Service Layer.
- 4. Domain Model:
- Components: Java classes representing entities (e.g., Book, Author)
- Purpose: Represents the core data and relationships of the application.
- Connections: Used by the Data Access Layer for ORM mapping

- 5. Configuration:
- Components: Application context configurations (e.g., @Configuration classes)
- Purpose: Configures beans and other aspects of the application.
- Connections: Provides settings for various layers and components.
- 6. Security Layer:
- Components: Spring Security configurations
- Purpose: Handles authentication and authorization, ensuring secure access to the application.
- Connections: May interact with the Presentation and Service Layers for access control.
- 7. Logging and Monitoring:
- Components: Logging frameworks, Spring Boot Actuator
- Purpose: Tracks application behavior and errors, and provides monitoring data.
- Connections: Monitors various layers for performance and health.

- 8. Exception Handling:
- Components: Exception handling mechanisms (e.g., @ControllerAdvice)
- Purpose: Manages exceptions and errors, and returns appropriate responses to the client.
- Connections: Interacts with the Presentation Layer and other layers.