UML Design Plan for Library Management System

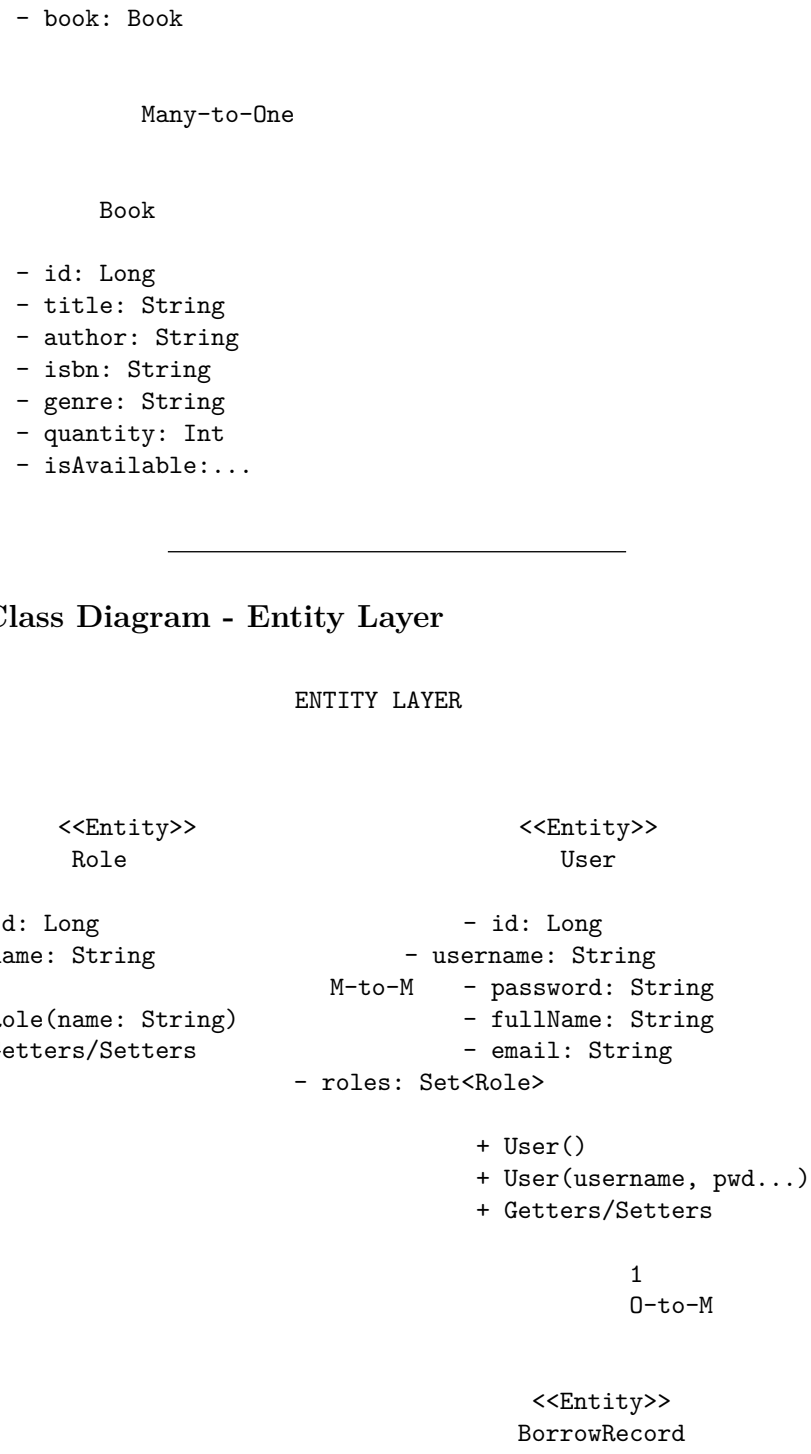
Project Overview

This is a **Spring Boot Library Management System** built with: - **Framework**: Spring Boot, Spring MVC, Spring Security, Spring Data JPA - **Database**: Relational Database (Flyway migrations) - **UI**: Thymeleaf Templates - **Architecture**: Three-layer architecture (Controller → Service → Repository)

---

1. Entity Relationship Diagram (ERD)

```
        Role

- id: Long
- name: String


           Many-to-Many


      User

- id: Long
- username: String
- password: String
- fullName: String
- email: String
- roles: Set<Role>

             One-to-Many


     BorrowRecord

    - id: Long
    - borrowDate
    - returnDeadline
    - actualReturnDate
    - status: String
    - user: User
```



---

2. Class Diagram - Entity Layer

```
- id: Long
- user: User
- book: Book
- borrowDate: LocalDate
- returnDeadline: LocalDate
- actualReturnDate: ...
- status: String

+ BorrowRecord()
+ BorrowRecord(user...)
+ Getters/Setters


          M-to-O


    <<Entity>>
       Book

- id: Long
- title: String
- author: String
- isbn: String (Unique)
- genre: String
- quantity: Integer
- isAvailable: Boolean

+ Book()
+ Book(title, author...)
+ Getters/Setters
```

---

3. Class Diagram - Repository Layer

```
REPOSITORY LAYER
(Spring Data JPA Repositories)


<<Interface>>                    <<Interface>>
 UserRepository                   BookRepository
```

```
extends CrudRepository<User>
+ findByUsername(): Optional
+ findByEmail(): Optional
+ existsByUsername(): boolean
+ existsByEmail(): boolean
```

```
extends CrudRepository<Book>
+ findByIsAvailable():List
+ findByGenre(): List
+ findByAuthor(): List
+ findByIsbn(): Optional
```

```
  <<Interface>>
  RoleRepository
extends CrudRepository<Role>
+ findByName(): Optional
```

```
  <<Interface>>
 BorrowRepository
extends CrudRepository<...>
+ findByUser(): List
+ findByStatus(): List
+ findActiveRecords(): List
```

---

4. Class Diagram - Service Layer

```
                    SERVICE LAYER
          (Business Logic & Transaction Management)
```

```
      <<Service>>
     BookService
- bookRepository: BookRepository
+ getAllBooks(): List<Book>
+ getBookById(id): Optional
+ getAvailableBooks(): List
+ saveBook(book): Book
+ updateBook(id, details): Book
+ deleteBook(id): void
+ searchBooks(keyword): List
+ getBooksByGenre(genre): List
```

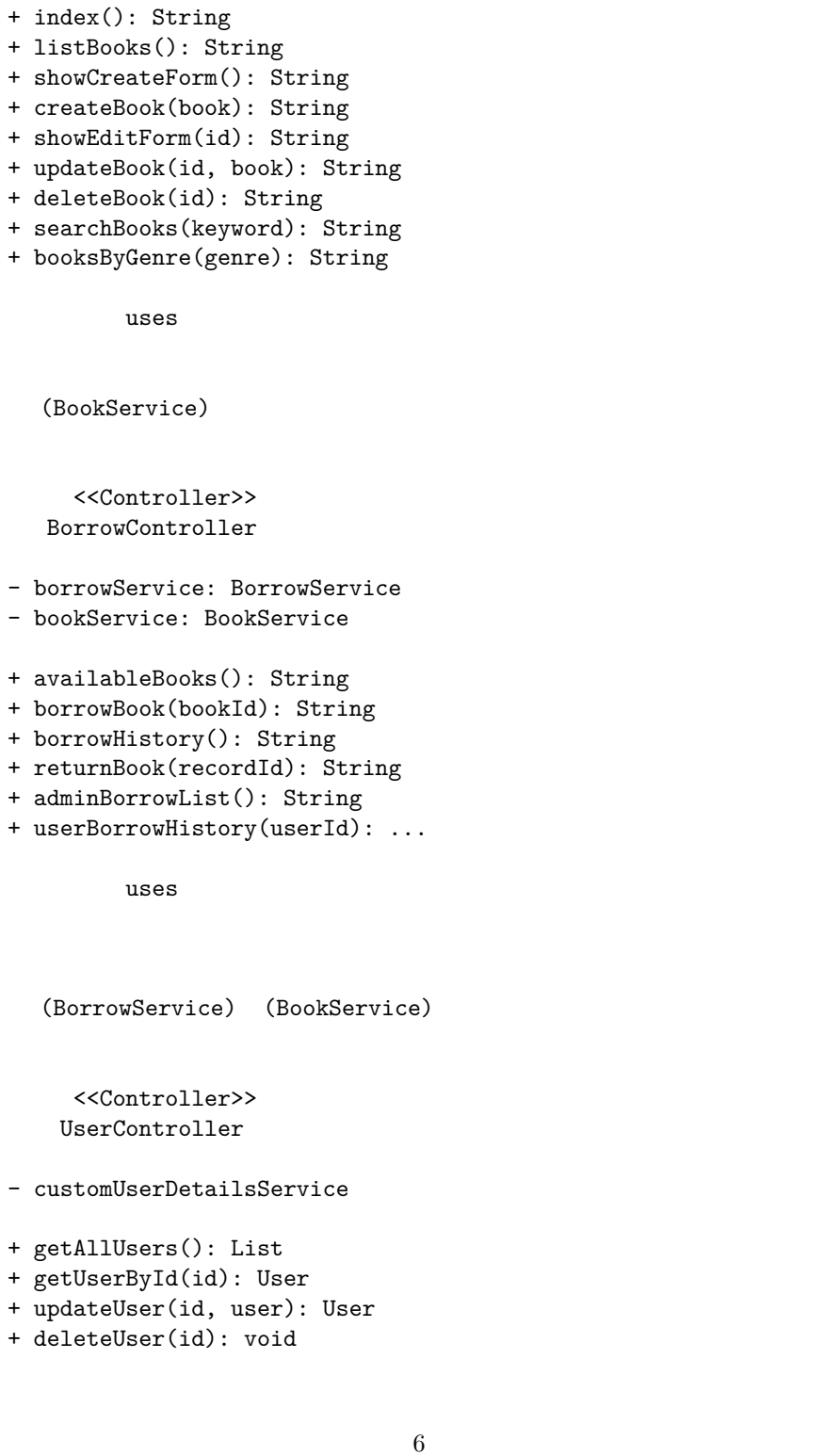
```
        <<Service>>                                  <<Service>>
```

```
      BorrowService                              CustomUserDetailsService

- borrowRepository: BorrowRepository          - userRepository: ...
- bookService: BookService
                              + loadUserByUsername():
+ borrowBook(user, book): BorrowRec             UserDetails
+ returnBook(recordId): BorrowRec            + loadUserById(id):
+ getBorrowHistory(userId): List                UserDetails
+ getActiveBorrows(userId): List
+ getOverdueBooks(): List
+ checkOverdueStatus(): void
+ getAdminBorrowList(): List

         uses


   (BookService)
```

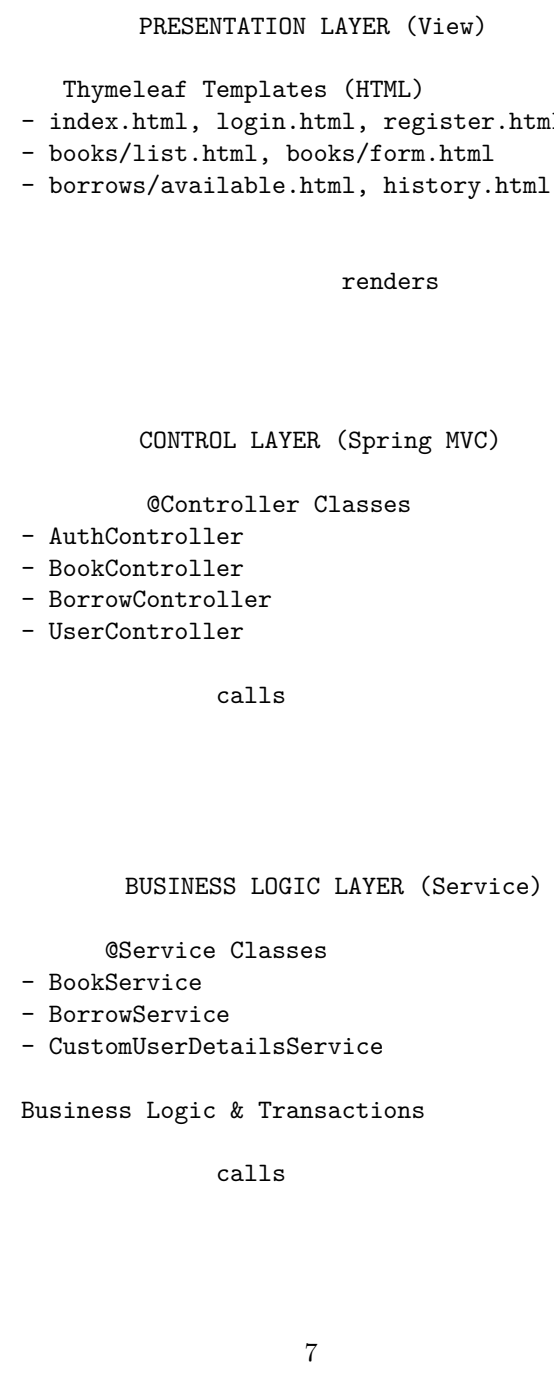
---

5. Class Diagram - Controller Layer

```
                    CONTROLLER LAYER
              (HTTP Request Handlers & MVC Flow)



     <<Controller>>
    AuthController

- customUserDetailsService

+ showLoginForm(): String
+ login(): String
+ showRegisterForm(): String
+ register(user): String
+ logout(): String



     <<Controller>>
    BookController

- bookService: BookService
```


+ index(): String
+ listBooks(): String
+ showCreateForm(): String
+ createBook(book): String
+ showEditForm(id): String
+ updateBook(id, book): String
+ deleteBook(id): String
+ searchBooks(keyword): String
+ booksByGenre(genre): String
uses
(BookService)
<<Controller>>
BorrowController
- borrowService: BorrowService
- bookService: BookService
+ availableBooks(): String
+ borrowBook(bookId): String
+ borrowHistory(): String
+ returnBook(recordId): String
+ adminBorrowList(): String
+ userBorrowHistory(userId): ...
uses
(BorrowService) (BookService)
<<Controller>>
UserController
- customUserDetailsService
+ getAllUsers(): List
+ getUserById(id): User
+ updateUser(id, user): User
+ deleteUser(id): void

6. Complete Three-Layer Architecture Diagram



```
DATA ACCESS LAYER (Repository)

Spring Data JPA Repositories
- UserRepository
- BookRepository
- BorrowRepository
- RoleRepository

(CRUD Operations & Custom Queries)

              accesses




DATABASE LAYER (Persistence)

   Relational Database (via JPA/Hibernate)

Tables:
- users (id, username, password, email)
- roles (id, name)
- user_roles (user_id, role_id)
- books (id, title, author, isbn, genre..)
- borrow_records (id, user_id, book_id..)

Schema initialized via Flyway migrations
```

---

7. Relationships Summary

Many-to-One Relationships

- **BorrowRecord** → **User** (Multiple borrow records belong to one user)
- **BorrowRecord** → **Book** (Multiple borrow records can reference one book)

Many-to-Many Relationships

- **User Role** (Users can have multiple roles, roles can be assigned to multiple users)

Inheritance

- No inheritance relationships (flat entity structure)

---

8. Security Architecture

```
      Spring Security Configuration

 - SecurityConfig.java
 - Custom UserDetailsService
 - Role-based Access Control (RBAC)
   • ROLE_ADMIN - Full access
   • ROLE_USER - Limited access

 Protected Endpoints:
 - /admin/* → ROLE_ADMIN only
 - /borrows/* → Authenticated users
 - /login → Public
 - /register → Public
```

---

9. Database Schema (Normalized)

```sql
-- Users Table
CREATE TABLE users (
    id BIGINT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);

-- Roles Table
CREATE TABLE roles (
    id BIGINT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);
```

```sql
-- User-Role Mapping (Many-to-Many)
CREATE TABLE user_roles (
    user_id BIGINT,
    role_id BIGINT,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (role_id) REFERENCES roles(id),
    PRIMARY KEY (user_id, role_id)
);

-- Books Table
CREATE TABLE books (
    id BIGINT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    isbn VARCHAR(20) UNIQUE NOT NULL,
    genre VARCHAR(100) NOT NULL,
    quantity INTEGER NOT NULL,
    is_available BOOLEAN NOT NULL
);

-- Borrow Records Table
CREATE TABLE borrow_records (
    id BIGINT PRIMARY KEY,
    user_id BIGINT NOT NULL,
    book_id BIGINT NOT NULL,
    borrow_date DATE NOT NULL,
    return_deadline DATE NOT NULL,
    actual_return_date DATE,
    status VARCHAR(20) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (book_id) REFERENCES books(id)
);
```

---

10. Key Design Patterns Used

Pattern	Usage	Location
MVC	Separates Model, View, Controller	Spring MVC Architecture
Repository Pattern	Abstract data access logic	Repository interfaces
Service Pattern	Encapsulates business logic	Service classes

Pattern	Usage	Location
Dependency Injection	Loose coupling via @Autowired	All layers
Singleton	Spring beans are singletons	@Service, @Repository
Template Method	JPA inheritance	CrudRepository implementations
DAO	Data Access Objects	Repository implementations

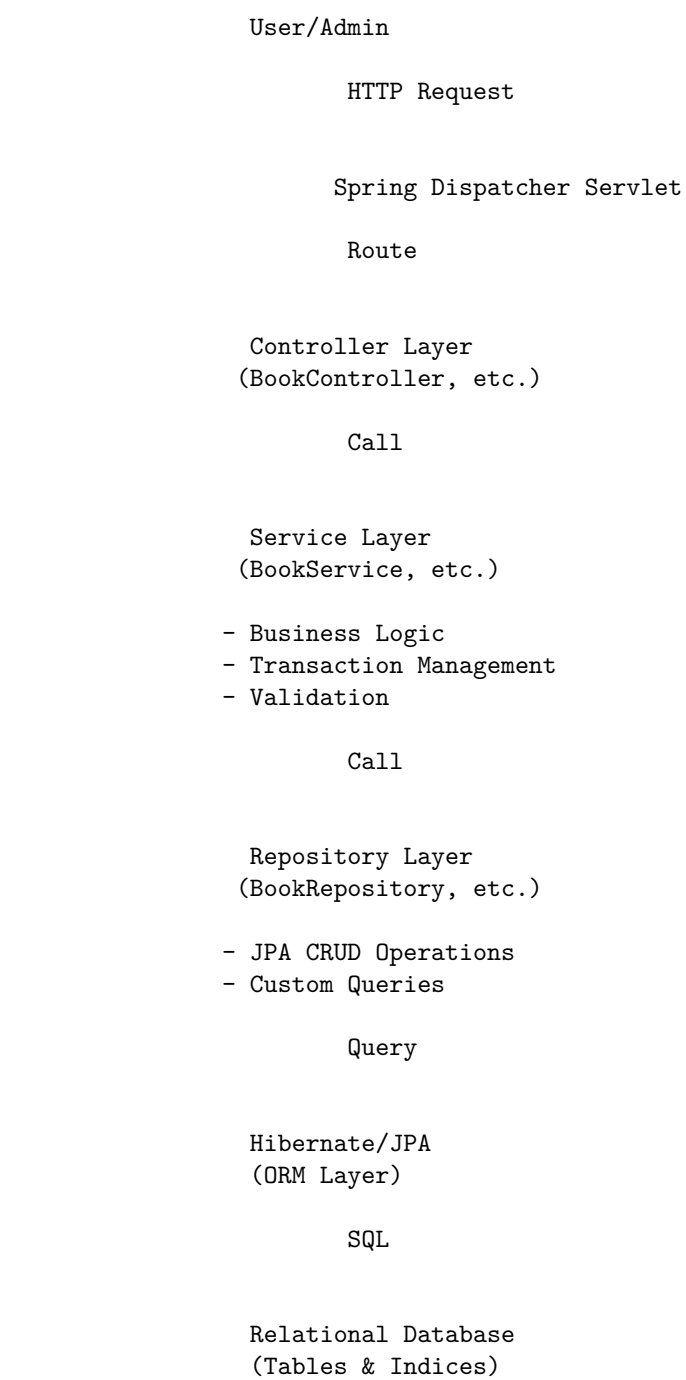
---

11. Data Flow Example - Borrowing a Book

```
User Request


[BorrowController.borrowBook(bookId)]

       Validates user is authenticated


[BorrowService.borrowBook(user, book)]

       Check if book is available

       Update book availability

       Create BorrowRecord entity
            - user: User
            - book: Book
            - borrowDate: LocalDate.now()
            - returnDeadline: borrowDate + 14 days
            - status: "BORROWED"


[BorrowRepository.save(borrowRecord)]


[Database - INSERT into borrow_records]


Return to View (redirect)
```

---

12. Component Interaction Diagram



Summary

This Library Management System follows a **three-layer architecture** (Controller → Service → Repository) with: - **4 Entity Classes**: User, Role, Book, BorrowRecord - **4 Repository Interfaces**: UserRepository, BookRepository, BorrowRepository, RoleRepository - **3 Service Classes**: BookService, BorrowService, CustomUserDetailsService - **4 Controller Classes**: AuthController, BookController, BorrowController, UserController - **Security**: Spring Security with role-based access control - **Database**: Relational database with Flyway migrations - **UI**: Thymeleaf templates with MVC pattern

The design emphasizes **separation of concerns**, **maintainability**, and **scalability**.