Detailed Design Document

Online Book Rental System – RESTful API

1. Introduction

This document outlines the detailed design of a RESTful API built with Spring Boot to manage an Online Book Rental System. The system supports:

- User registration and authentication
- Book and rental management
- Role-based access control

Data is persisted in a MySQL database.

2. System Architecture

2.1 Overview

The system adopts a layered architecture:

- Controller Layer: Handles HTTP requests, responses, and validation.
- Service Layer: Contains business logic.
- Repository Layer: Communicates with MySQL using Spring Data JPA.
- Security Layer: Handles authentication and authorization using Spring Security with Basic Auth.
- Model Layer: Defines entities (User, Book, Rental).

2.2 Technology Stack

Java 17+
Spring Boot
Spring Security
Spring Data JPA
MySQL
BCrypt
Maven/Gradle
JUnit, Mockito
SLF4J/Logback

3. Database Design

3.1 ERD

User (1) --- (N) Rental (N) --- (1) Book

3.2 Tables and Fields

- users: id, email, password, first_name, last_name, role
- books: id, title, author, genre, availability_status
- rentals: id, user_id, book_id, rental_date, return_date

4. Security Design

4.1 Authentication

Uses HTTP Basic Authentication. Passwords hashed using BCrypt.

4.2 Authorization

Roles: USER and ADMIN. Public endpoints (e.g., registration) are open. Private endpoints are protected by roles.

Example Spring Security Configuration:

```
.antMatchers("/api/auth/**").permitAll()
.antMatchers(HttpMethod.GET, "/api/books/**").hasAnyRole("USER", "ADMIN")
.antMatchers("/api/books/**").hasRole("ADMIN")
.antMatchers("/api/rentals/**").hasRole("USER")
```

5. API Design

5.1 Endpoint Summary

- POST /api/auth/register – Public
- GET /api/books – USER, ADMIN
- POST/PUT/DELETE /api/books – ADMIN only
- POST /api/rentals/rent/{bookId} – USER only
- POST /api/rentals/return/{bookId} – USER only

5.2 Sample Requests and Responses

User Registration:
POST /api/auth/register

```
{
  "email": "user@example.com",
  "password": "password123",
  "firstName": "John",
  "lastName": "Doe",
```

"role": "USER"

}

Get Books:
GET /api/books
> Requires Basic Auth

6. Business Logic

- Users are registered with hashed passwords and a default role of USER.
- ADMIN users can manage book data.
- USER can rent a maximum of 2 books concurrently and return them when done.

7. Error Handling Strategy

Uses @ControllerAdvice to manage exceptions:
- 404 – User, Book, or Rental not found
- 400 – Validation or business logic errors
- 403 – Unauthorized access

8. Logging

Uses SLF4J with Logback to log:
- INFO
- WARN
- ERROR

9. Testing Strategy

Tools: JUnit, Mockito, MockMvc

Key Test Cases:
1. User registration and password hashing
2. Fetching books
3. Rental limit enforcement

10. Example Code Snippets

Password Hashing (UserService):

String hashedPassword = passwordEncoder.encode(plainPassword);
user.setPassword(hashedPassword);

Global Exception Handler (@RestControllerAdvice):

```
@ExceptionHandler(ResourceNotFoundException.class)
 public ResponseEntity<ErrorResponse>
handleNotFound(ResourceNotFoundException ex) {
   return new ResponseEntity<>(new ErrorResponse(ex.getMessage()),
HttpStatus.NOT_FOUND);
 }
```