**Assignment 14**

**Name: Sahil Kumar**

**Roll No : 13000116068**

**Problem Statement** Design and implement the following requirements under a typical Library Management System:

R1. Any library member should be able to search books by their title, author, subject category as well by the publication date.  
R2. Each book will have a unique identification number and other details including a rack number which will help to physically locate the book.  
R3. There could be more than one copy of a book, and library members should be able to check-out and reserve any copy. We will call each copy of a book, a book item.  
R4. The system should be able to retrieve information like who took a particular book or what are the books checked-out by a specific library member.  
R5. There should be a maximum limit (5) on how many books a member can check-out.  
R6. There should be a maximum limit (10) on how many days a member can keep a book.  
R7. The system should be able to collect fines for books returned after the due date.  
R8. Members should be able to reserve books that are not currently available.

Top use cases of the Library Management System:

**1. Add/Remove/Edit book:** To add, remove or modify a book or book item.  Bulk addition of books and users should be allowed.  
**2. Register new account/cancel membership:** To add a new member or cancel the membership of an existing member.  
**3. Search catalog:** To search books by title, author, subject or publication date.  
**4. Issue book:** To borrow a book from the library.  
**5. Return a book:** To return a book to the library which was issued to a member.  
**6. Reserve book:** To reserve a book which is not currently available

**7. Renew a book:** To reborrow an already checked-out book

**Software And Tools**

**OS** Mac OS

Programming language JAVA, NetBeans Ide, mysql

**Algorithm**

1. Start
2. User can register themselves in the application
3. The user can issue a book from the library
4. If the user is late they will be fined for being late
5. Separate database is created for users and book
6. Stop

**Code**

**Library Controller**

package darkarmy.springframework.librarymanagementsystem.controller;

import darkarmy.springframework.librarymanagementsystem.model.Book;

import darkarmy.springframework.librarymanagementsystem.model.BorrowedBook;

import darkarmy.springframework.librarymanagementsystem.model.ReservedBook;

import darkarmy.springframework.librarymanagementsystem.model.User;

import darkarmy.springframework.librarymanagementsystem.repository.BookRepository;

import darkarmy.springframework.librarymanagementsystem.repository.BorrowedBookRepository;

import darkarmy.springframework.librarymanagementsystem.repository.ReservedBookRepository;

import darkarmy.springframework.librarymanagementsystem.repository.UserRepository;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.time.LocalDate;

import java.util.Map;

@RestController

@RequestMapping("/api/library")

public class LibraryController {

private UserRepository userRepository;

private BookRepository bookRepository;

private BorrowedBookRepository borrowedBookRepository;

private ReservedBookRepository reservedBookRepository;

public LibraryController(UserRepository userRepository, BookRepository bookRepository, BorrowedBookRepository borrowedBookRepository, ReservedBookRepository reservedBookRepository) {

this.userRepository = userRepository;

this.bookRepository = bookRepository;

this.borrowedBookRepository = borrowedBookRepository;

this.reservedBookRepository = reservedBookRepository;

}

// issue book

@PostMapping("/issue")

public ResponseEntity<?> issueBook(@RequestBody Map<String, String> map){

Long userId = Long.valueOf(map.get("userId"));

Long bookId = Long.valueOf(map.get("bookId"));

LocalDate tillDateBorrow = LocalDate.parse(map.get("date"));

Book book = bookRepository.findById(Math.toIntExact(bookId)).get();

User user = userRepository.findById(Math.toIntExact(userId)).get();

if(book.isBorrowed() || book.isReserved()){

return new ResponseEntity<>("Book "+book.getTitle()+ "is not available.", HttpStatus.BAD\_GATEWAY);

}

if(user.getBorrowedBooks().size()>=5){

return new ResponseEntity<>("User "+user.getFirstName()+ " already borrowed five books.", HttpStatus.BAD\_GATEWAY);

}

if(tillDateBorrow.compareTo(LocalDate.now())>=10){

return new ResponseEntity<>("You can't borrow book more then 10 days.", HttpStatus.BAD\_GATEWAY);

}

book.setBorrowed(true);

BorrowedBook borrowedBook = borrowedBookRepository.save(new BorrowedBook(user, book,tillDateBorrow));

return new ResponseEntity<>(borrowedBook, HttpStatus.CREATED);

}

// return book

@PostMapping("/return")

public ResponseEntity<?> returnBook(@RequestBody Map<String, String> map){

Long userId = Long.valueOf(map.get("userId"));

Long bookId = Long.valueOf(map.get("bookId"));

Book book = bookRepository.findById(Math.toIntExact(bookId)).get();

User user = userRepository.findById(Math.toIntExact(userId)).get();

BorrowedBook borrowedBook = borrowedBookRepository.findByUserAndBook(user, book);

LocalDate date = borrowedBook.getTillDateToBorrow();

Integer days;

if(-(days=date.compareTo(LocalDate.now()))>0){

user.setFine(-days);

}

borrowedBookRepository.deleteById(borrowedBook.getId());

return new ResponseEntity<>("Book: "+book.getTitle()+"returned successfully.", HttpStatus.OK);

}

// reserve book

@PostMapping("/reserve")

public ResponseEntity<?> reserveBook(@RequestBody Map<String, String> map){

Long userId = Long.valueOf(map.get("userId"));

Long bookId = Long.valueOf(map.get("bookId"));

Book book = bookRepository.findById(Math.toIntExact(bookId)).get();

User user = userRepository.findById(Math.toIntExact(userId)).get();

if(book.isReserved()){

return new ResponseEntity<>("Book: "+book.getTitle()+" id already reserved.", HttpStatus.BAD\_REQUEST);

}

ReservedBook reservedBook = reservedBookRepository.save(new ReservedBook(user, book));

return new ResponseEntity<>("Book: "+book.getTitle()+"returned successfully.", HttpStatus.OK);

}

// renew book

@PutMapping("/return")

public ResponseEntity<?> renewBook(@RequestBody Map<String, String> map){

Long userId = Long.valueOf(map.get("userId"));

Long bookId = Long.valueOf(map.get("bookId"));

LocalDate tillDateBorrow = LocalDate.parse(map.get("date"));

Book book = bookRepository.findById(Math.toIntExact(bookId)).get();

User user = userRepository.findById(Math.toIntExact(userId)).get();

BorrowedBook borrowedBook = borrowedBookRepository.findByUserAndBook(user, book);

if(borrowedBook==null){

return new ResponseEntity<>("Book: "+book.getTitle()+" is not even issued.", HttpStatus.BAD\_GATEWAY);

}

borrowedBook.setTillDateToBorrow(tillDateBorrow);

borrowedBookRepository.save(borrowedBook);

return new ResponseEntity<>("Book: "+book.getTitle()+"renewed successfully.", HttpStatus.OK);

}

}

**Book Controller**

package darkarmy.springframework.librarymanagementsystem.controller;

import darkarmy.springframework.librarymanagementsystem.model.Book;

import darkarmy.springframework.librarymanagementsystem.repository.BookRepository;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/book")

public class BookController {

private BookRepository bookRepository;

public BookController(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

@GetMapping("/all")

@ResponseStatus(HttpStatus.OK)

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

@ResponseStatus(HttpStatus.OK)

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(Math.toIntExact(id)).get();

}

@GetMapping("/search/title/{searchText}")

@ResponseStatus(HttpStatus.OK)

public Book getByTitle(@PathVariable String searchText) {

return bookRepository.findByTitle(searchText);

}

@GetMapping("/search/author/{searchText}")

@ResponseStatus(HttpStatus.OK)

public List<Book> getByAuthor(@PathVariable String searchText) {

return bookRepository.findByAuthor(searchText);

}

@GetMapping("/search/publication/{searchText}")

@ResponseStatus(HttpStatus.OK)

public List<Book> getByPublicationDate(@PathVariable String searchText) {

return bookRepository.findByPublicationDate(searchText);

}

@GetMapping("/search/subject/{searchText}")

@ResponseStatus(HttpStatus.OK)

public List<Book> getBySubject(@PathVariable String searchText) {

return bookRepository.findBySubject(searchText);

}

@PostMapping("/add")

@ResponseStatus(HttpStatus.CREATED)

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/update/{id}")

@ResponseStatus(HttpStatus.OK)

public String updateBook(@PathVariable Integer id, @RequestBody Book book) {

Optional<Book> book1 = bookRepository.findById(id);

if (book1 != null) {

bookRepository.deleteById(id);

bookRepository.save(book);

return "Book " + book.getTitle() + " updated successfully! " + book.toString();

} else {

return "Invalid book id";

}

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.OK)

public String deleteBook(@PathVariable Integer id) {

bookRepository.deleteById(id);

return "Book with id " + id + " deleted successfully!";

}

}

**User**

package darkarmy.springframework.librarymanagementsystem.controller;

import darkarmy.springframework.librarymanagementsystem.model.User;

import darkarmy.springframework.librarymanagementsystem.repository.UserRepository;

import org.jasypt.util.password.StrongPasswordEncryptor;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/user")

public class UserController {

private UserRepository userRepository;

private StrongPasswordEncryptor strongPasswordEncryptor;

public UserController(UserRepository userRepository, StrongPasswordEncryptor strongPasswordEncryptor) {

this.userRepository = userRepository;

this.strongPasswordEncryptor = strongPasswordEncryptor;

}

@GetMapping("/all")

@ResponseStatus(HttpStatus.OK)

public List<User> getAllBooks() {

return userRepository.findAll();

}

@GetMapping("/{id}")

@ResponseStatus(HttpStatus.OK)

public Optional<User> getUserById(@PathVariable Integer id) {

return userRepository.findById(id);

}

@PostMapping("/add")

@ResponseStatus(HttpStatus.CREATED)

public User addUser(@RequestBody User user) {

String password = strongPasswordEncryptor.encryptPassword(user.getPassword());

return userRepository.save(new User(user.getId(), user.getFirstName(), user.getLastName(), user.getGender(), user.getEmail(), user.getUserName(), password));

}

@PostMapping("/login")

@ResponseStatus(HttpStatus.OK)

public String loginUser(@RequestBody User user) {

String password = user.getPassword();

String userName = user.getUserName();

User user1 = userRepository.findByUserName(userName);

if (user1 != null) {

if (strongPasswordEncryptor.checkPassword(password, user1.getPassword())) {

return "User " + user1.getUserName() + " logged in successfully! \n" + user1.toString();

} else {

return "Password is wrong!";

}

} else {

return "Invalid User";

}

}

@PutMapping("/update/{userName}")

@ResponseStatus(HttpStatus.OK)

public String updateUser(@RequestBody User user) {

String userName = user.getUserName();

User user1 = userRepository.findByUserName(userName);

if (user1 != null) {

userRepository.delete(user1);

userRepository.save(user);

return "User updated successfully /n" + user.toString();

} else {

return "Invalid User";

}

}

@DeleteMapping("/{id}")

@ResponseStatus(HttpStatus.OK)

public String deleteUser(@PathVariable Integer id) {

userRepository.deleteById(id);

return "User with id " + id + " deleted successfully!";

}

}

**Output**

**Users table**

A screenshot of a cell phone

Description automatically generated

**Book Table**

A screenshot of a cell phone

Description automatically generated

**Borrowed Book**A screenshot of a cell phone

Description automatically generated