DOKUZ EYLÜL UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CME 2210

Object Oriented Analysis and Design

LIBRARY MANAGEMENT SYSTEM

by

Nuri Çilengir 2016510111

Hasan Berk Kocabaş 2016510046

Mustafa Deniz 2016510018

CHAPTER ONE
INTRODUCTION

1.1 Subject

The subject of the project is library management. Library management system will designed to digitize and simplfy the library management.

1.2 Purpose

The purpose of that document is to present a description of the Library Management System. This document explain the all details this software project that database design, operations, interfaces and use of the application.

1.3 Scope

Library Management System is a desktop base application for students, teachers and managers. The system will allow universities to manage their libraries easily with user friendly interfaces and system tools.

More specifically, the LibraryManagementSystem will also allow students to search for books and request them. Teachers also can search books, request them and add publication to system. Managers can edit all informations about users, books and respond to book requests.

1.4 References

[1] IEEE Software Engineering Standards Committee, "IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications", October 20, 1998

CHAPTER TWO
REQUIREMENTS

2.1 External Interfaces

Registiration Screen: That page is for the user registiration.

Login Screen: That page is allow to user and manager login.

Profile Page: That page is allow the show user informations and edit them. In this page also show user activity history.

Collection Page: In this page books are listed. User can search and request for available book.

Manager Page: Manager can manage all users, books and request in this page.

Report page: In this page all reports are listed. For example: book requests and logs.

Apply Page: For the users allow the book request. Users can see available date interval to the take book. (In the modal)

2.2 Functions

Crud Operations: Users and books can be deleted, updated, created and readed.

Search Operations: Collections, authors, publishers, users can be searched.

Filter Operations: Category, author, publisher, publication date filter can be applied.

Request Operation: User can request for a book.

Calculation Operations: Total user, total book, total request, success – unsuccess requests can be calculated.

Book isAvailable function: This function show to users that book is available or not.

2.3 Performance Requirements

We will use hibernate ecosystem for ORM operations. The software should be capable of storing all the data so arraylist, stack and queue structures will be used.

2.4 Logical Database Requirements

All data that user accounts, collections, etc. will be saved in the database. Program requires a fast and simple design to reliability and maintainability. Hibernate ecosystem and mysql will be used.

2.5 Design Constraints

User Interfaces will be written in Java Web Application. N-Tier architecture will be used for reusable and systematic application. React will be used for web frontend.

2.6 Software System Quality Attributes

The program must be reliable, support new versions, maintainable for necessary updates and security is the other important thing for Library Management System. Spring security and
JVT ecosystem will be used for authentication. Test driven development will be used with Junit.

2.7 Analysis Class Model

Class Author: name

Class Book: title, authors, category_ıd, publısher_ıd, type, isbn, edition, publication_year, description

Class Entry: book_ıd, user_ıd, status, request_date, return_date, return_status

Class BookInfo: book_ıd, return_date, availability

Class Category: name

Class Publisher: name

Class User: name, email, password, city_id, phone, address, zipcode, sex, bdate, isadmin

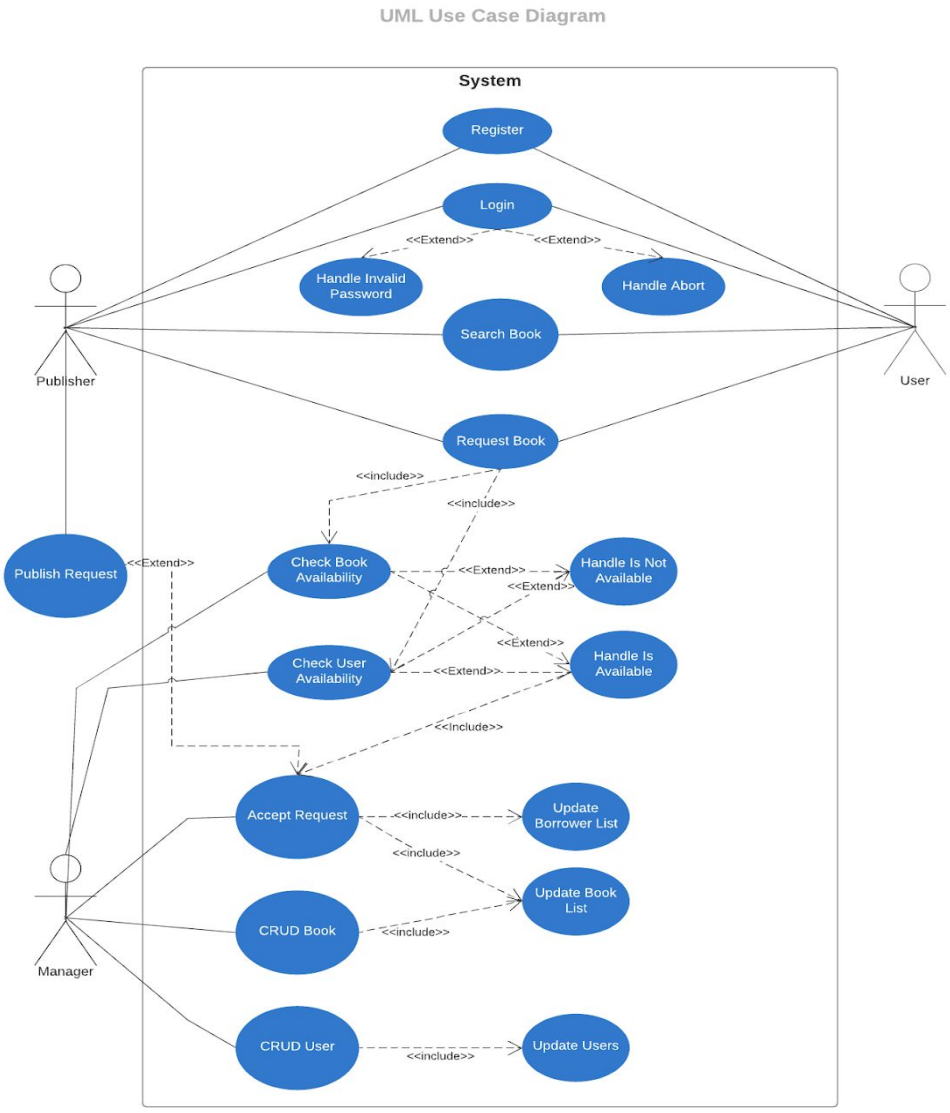
Class city: name

Class genre: id, name, description

CHAPTER THREE

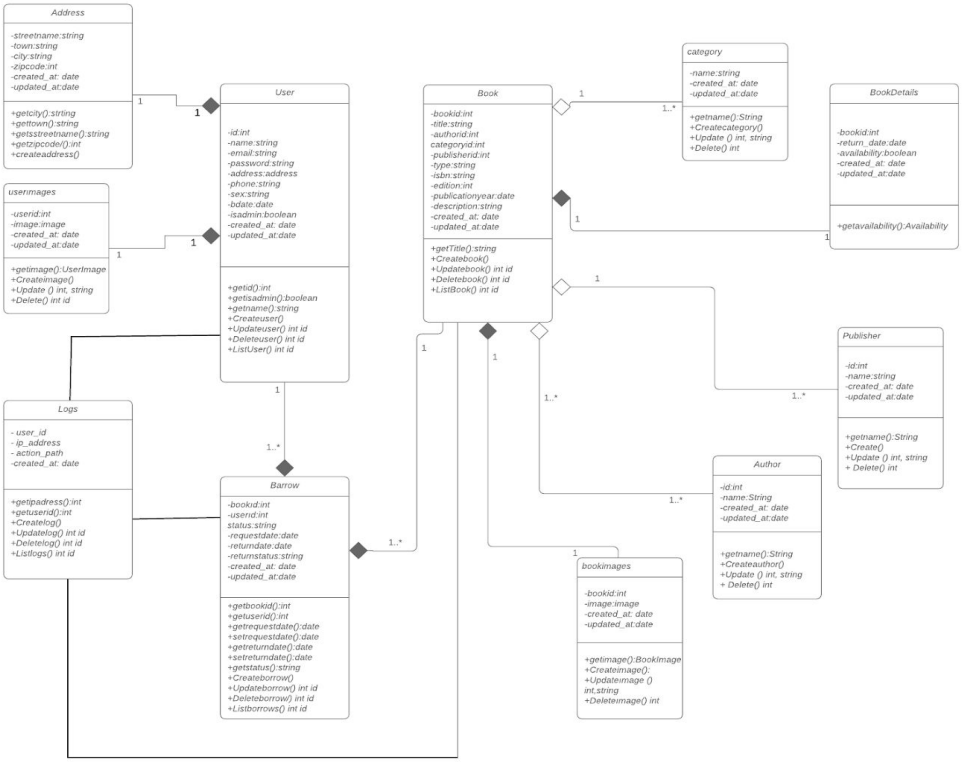
UML DIAGRAMS

3.1 Use Case Diagram



This diagram shows the abilities of the actors in the program in general.The program has 3 actors which are Manager, Standart User(Generally Students) and Publisher. Standart User has ability for registering, searching and requesting books. Publishers, in addition to Standart User they can publish publishment. Managers can access all managerials functions such as deleting, creating, updating and reading all data. Other manager task is the control requrests and accept or decline them. Managers perform this task based on book and user status.

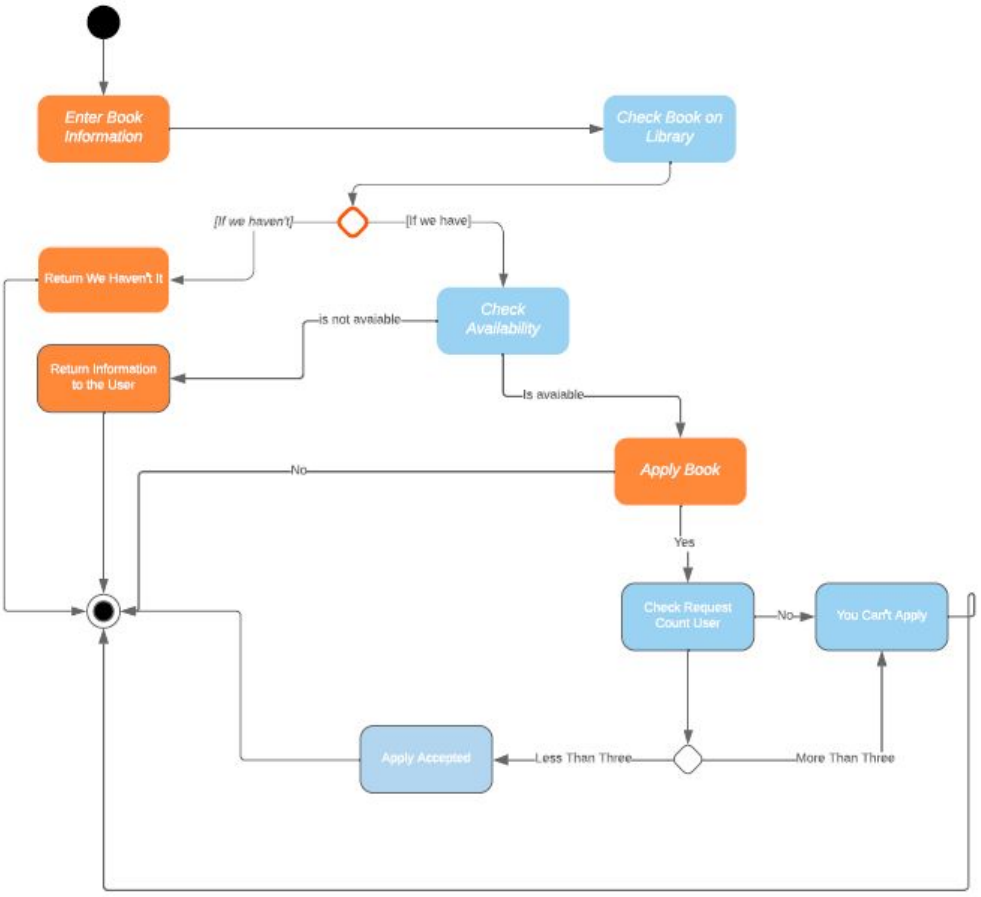
3.2 Class Diagram



Class diagram is listed above. We have user and book class like a main class. Barrow class connected user and book classes. All user types having address, ımage class and some attributes. Books also have category, details, publisher, author and images classes. Barrow

class connected user and book classes. Logs class keeping records belong to user and book operations.

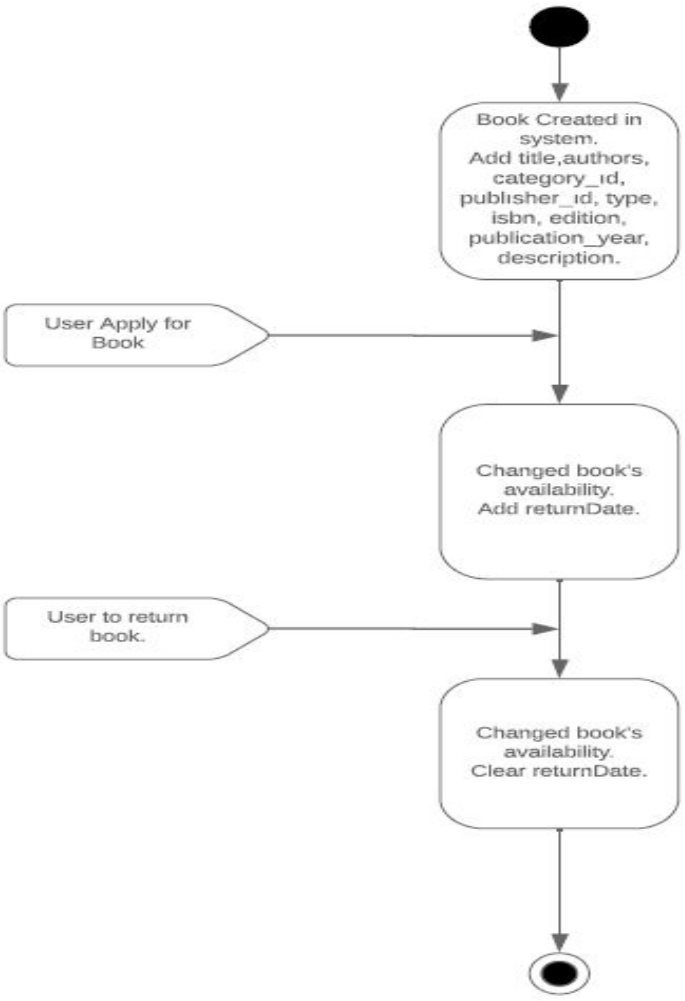
3.3 Activity Diagram



This diagram show us the path of books when hire. First time user enter a book then Book checked on the system if it can be borrow user can apply it but if it can't available system return the book isn't available. When user apply for book system checks request count of user.

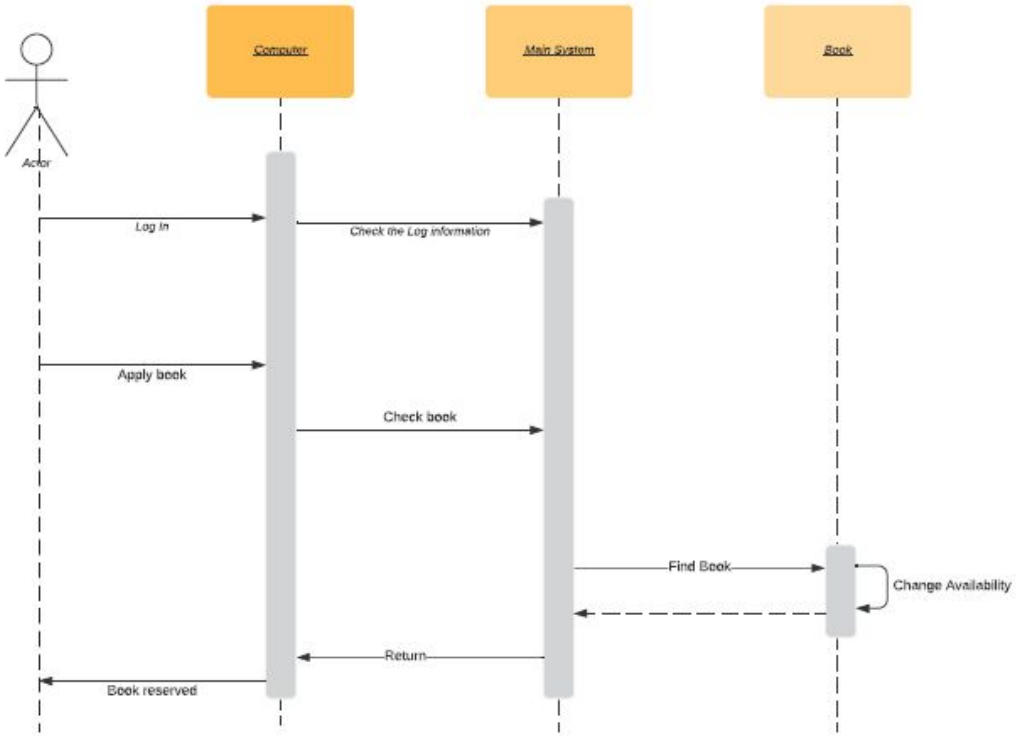
If it more than three user can't apply for borrow a book if less than three he can apply and borrow the book.

3.4 State Diagram



In this state diagram we tell book status. First time book created in system we add some information about book. Then when user apply for book book's availability status is change and system add a returnDate. When user to return the book system made book status available and clear the returnDate.

3.5 Sequence Diagram



In this diagram our objects ae computer main system and book. First action is logged in system by user then system check the information about the user ıf the information true user can apply for books. Second action is applying book. In this diagram we calculate best case scenario so user applied book and system confirm book in the system and return book can borrow. So computer return this information to user. The object book just change its availability.

CHAPTER FOUR

IMPLEMENTATIONS

1) Project Structure

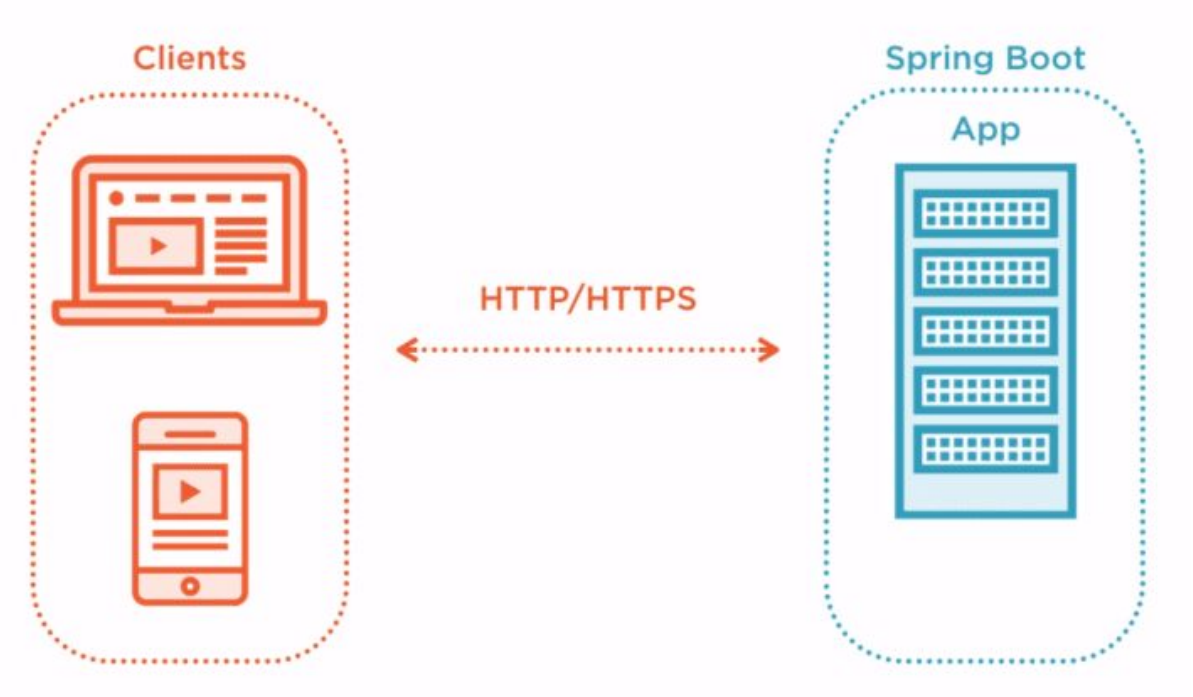
Database connections are indispensable in our projects, it will not be very accurate to load the entire load UI (user interface) when we want to perform operations such as pulling, adding, deleting, updating data from the database.

Data Transfer Object

DTO is an object that carries data between processes. When you're working with a remote interface, each call it is expensive. As a result you need to reduce the number of calls. The solution is to create a Data Transfer Object that can hold all the data for the call. It needs to be *serializable* to go across the connection. Usually an assembler is used on the server side to transfer data between the DTO and any domain objects. It's often little more than a bunch of fields and the getters and setters for them.

Service objects are doing the work that the application needs to do for the domain you're working with. It involves calculations based on inputs and stored data, validation of any data that comes in from the presentation, and figuring out exactly what data source logic to dispatch, depending on commands received from the presentation. A Service Layer defines an application's boundary and its set of available operations from the perspective of interfacing client layers. It encapsulates the application's business logic, controlling transactions and coordinating responses in the implementation of its operations.

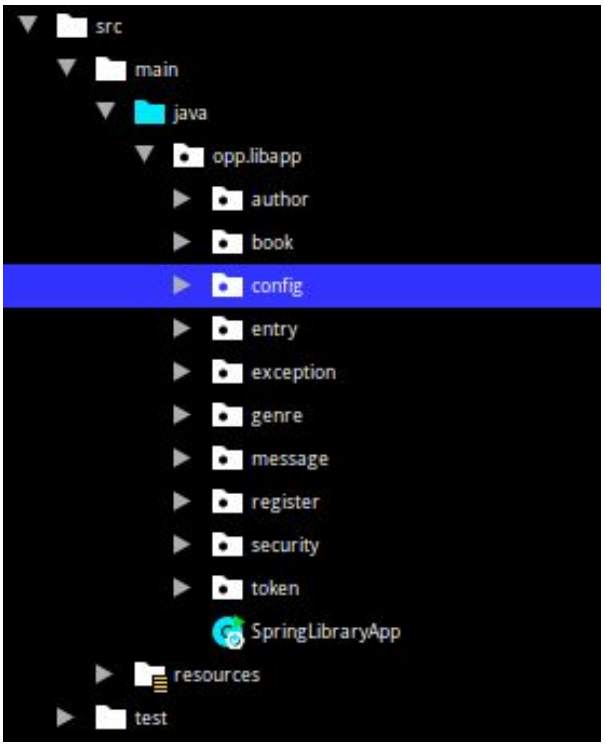
RESTful (Representational State Transfer) is an architectural design pattern. It describes that the different independent components of your application should communicate with each other through simple HTTP/HTTPS calls. If an application satisfies this criterion it can be considered as a RESTful web application.



In a RESTful web application, you should utilize the different HTTP verbs for the CRUD operations:

REST is the underlying architectural principle of the web. The main benefit of it is that the client and the server can not know anything about each other. Therefore, you can use any client or server technology, which can send, accept, and can respond to HTTP requests. This is the method that shows how the different microservices can communicate with each other in Spring Cloud.

After all TL;DL and descriptions, We look our project structure.


src
main
java
opp.libapp
author
book
config
entry
exception
genre
message
register
security
token
SpringLibraryApp
resources
test

2) Entities


entities
Author
Barrow
Book
BookDetail
BookImage
Category
City
Collection
Publishers
User

We have implements our previously mentioned classes in the their disctinct package. We have implemented the JAVA implementation because we have the idea to turn the project into an API-based one. To give an example, let's show our preme on the main scheme of the USER class.

First of all, we have implemented our attributes.

```java
package opp.libapp.register;

import ...

@Entity
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String username;

    @JsonIgnore
    private String password;

    @JsonIgnore
    private String secret;

    @JsonIgnore
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.PERSIST)
    @JoinTable(
            name="user_authorities",
            joinColumns = @JoinColumn(name="user_fk", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name="authority_fk", referencedColumnName = "id")
    )
    private Set<Authority> authorities;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getUsername() { return username; }
```

Then we added our constructor and getter / setter methods.

```
private Set<Authority> authorities;

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getUsername() { return username; }

public void setUsername(String username) { this.username = username; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }

public Set<Authority> getAuthorities() { return authorities; }

public void setAuthorities(Set<Authority> authorities) { this.authorities = authorities; }

public String getSecret() { return secret; }

public void setSecret(String secret) { this.secret = secret; }

@JsonIgnore
@Override
public boolean isAccountNonExpired() { return true; }

@JsonIgnore
@Override
public boolean isAccountNonLocked() { return true; }

@JsonIgnore
```

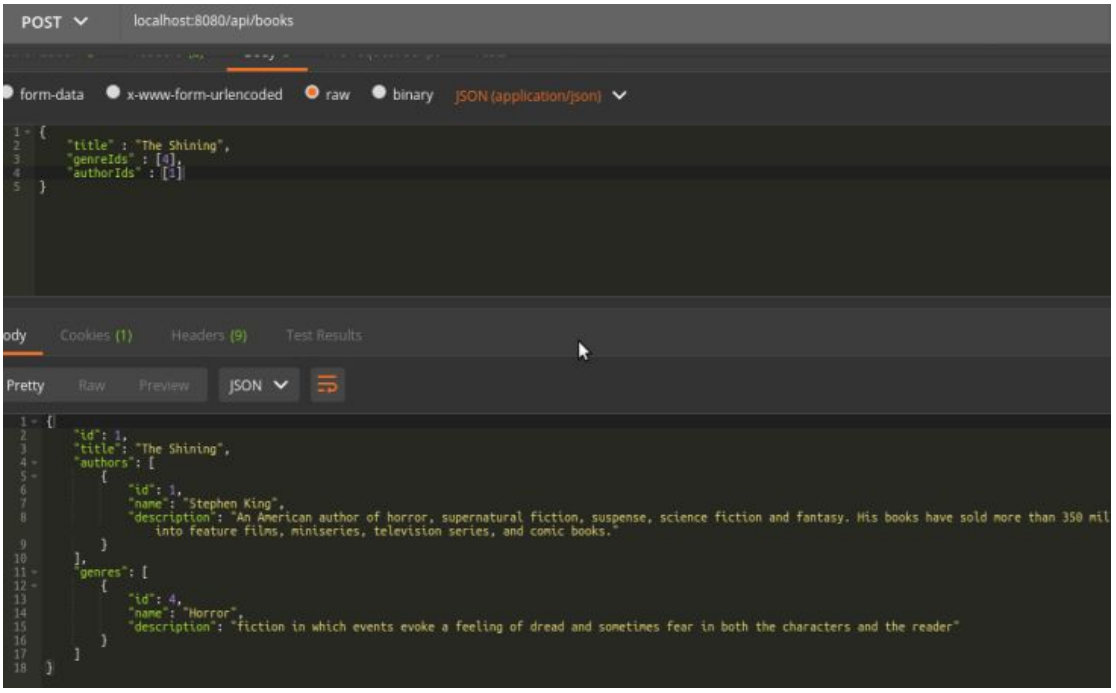
Some of classes;

Registration and token generation endpoints

Endpoint	Method	Data sent with the request	Description
/users/register	POST	JSON	On this endpoint new users can be registered
/users/get-token	GET	Basic authentication data must be provided	After successful authorization, token that is going to be valid for 3 days is generated and returned

Endpoint	Method	Data sent with the request	Description
/api/authors	GET		Display all existing authors.
/api/authors?name=	GET		Display authors filtered by name
/api/authors/{id}	GET		Display the author with specified id
/api/authors	POST	JSON	Create a new author
/api/authors/{id}	PUT	JSON	Replace old author data with new data
/api/authors/{id}	DELETE		Delete the author that has specified id
/api/genres	GET		Display all existing genres
/api/genres/{id}	GET		Display the genre with specified id
/api/genres	POST	JSON	Create a new genre
/api/genres/{id}	PUT	JSON	Replace old genre data with new data
/api/genres/{id}	DELETE		Delete genre that has specified id
/api/books	GET		Display all existing books
/api/books?title=	GET		Display books filtered by title
/api/books?author=	GET		Display books filtered by author name
/api/books?genre=	GET		Display books filtered by genre name
/api/books/{id}	GET		Display the book that has specified id
/api/books/{id}/bookInfo	GET		Display book info of the book that has specified id
/api/books/{id}/bookInfo	PATCH	JSON	Update chosen fields of book info
/api/books	POST	JSON	Create a new book
/api/books	PATCH	JSON	Update chosen fields of book
/api/books/{id}	DELETE		Delete book that has specified id
/api/entries	GET		Display all existing entries
/api/entries?returned=	GET		Display entries filtered by "returned status". This parameter accepts "true"/"false" values.
/api/entries?bookTitle=	GET		Display entries filtered by book title
/api/entries?username=	GET		Display entries filtered by **exact** username
/api/entries?since=	GET		Display entries added since specified time. Accepted values: *day*/*week*/*month*/*year*. If the parameter value does not belong to accepted values, all entries are going to be displayed
/api/entries/{id}	GET		Display entry that has specified id
/api/entries	POST	JSON	Create a new entry
/api/entries/{id}	PATCH	JSON	Update status of the entry that has specified id

Create book API response;



Get books API response;

GET localhost:8080/api/books

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "title": "The Shining",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
                "description": "An American author of horror, supernatural fiction, suspense, science fiction and fantasy. His books have
                    adapted into feature films, miniseries, television series, and comic books."
            }
        ],
        "genres": [
            {
                "id": 4,
                "name": "Horror",
                "description": "fiction in which events evoke a feeling of dread and sometimes fear in both the characters and the reader
            }
        ]
    },
    {
        "id": 2,
        "title": "It",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
                "description": "An American author of horror, supernatural fiction, suspense, science fiction and fantasy. His books have
                    adapted into feature films, miniseries, television series, and comic books."
            }
        ],
        "genres": [
            {
                "id": 4,
                "name": "Horror",
                "description": "fiction in which events evoke a feeling of dread and sometimes fear in both the characters and the reader
            }
        ]
    },
    {
        "id": 3,
        "title": "Pet Semetary",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
```


Filter book by filter API response;

GET localhost:8080/api/books?title=it

Pretty Raw Preview JSON


```
[
    {
        "id": 2,
        "title": "It",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
                "description": "An American author of horror, supernatural fiction, suspense, science fiction and fantasy. His books ha
                    adapted into feature films, miniseries, television series, and comic books."
            }
        ],
        "genres": [
            {
                "id": 4,
                "name": "Horror",
                "description": "fiction in which events evoke a feeling of dread and sometimes fear in both the characters and the rea
            }
        ]
    },
    {
        "id": 4,
        "title": "The Hobbit",
        "authors": [
            {
                "id": 3,
                "name": "J. R. R. Tolkien",
                "description": "An English writer, poet, philologist, and university professor who is best known as the author of the
                    The Silmarillion. While many other authors had published works of fantasy before Tolkien,[3] the great success of
                    resurgence of the genre. This has caused Tolkien to be popularly identified as the \"father\" of modern fantasy li
                    ranked him sixth on a list of \"The 50 greatest British writers since 1945\". Forbes ranked him the 5th top-earnin
            }
        ],
        "genres": [
            {
                "id": 5,
                "name": "Fantasy",
                "description": "fiction in a unreal setting that often includes magic, magical creatures, or the supernatural"
            }
        ]
    }
]
```

Stack Usage;

```java
import javax.validation.Valid;
import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

@RestController
public class AuthorController {

    private IAuthorService authorService;

    @Autowired
    public AuthorController(IAuthorService authorService) { this.authorService = authorService; }

    @RequestMapping(value = "api/authors", method = RequestMethod.GET)
    public ResponseEntity<Stack<Author>> getAuthors(@RequestParam(value = "name", required = false) String name) {
        Stack<Author> authors;
        if (name == null) {
            authors = (Stack<Author>) authorService.findAll();
        } else {
            authors = (Stack<Author>) authorService.findAllByNameContaining(name);
        }
        return new ResponseEntity<>(authors, HttpStatus.OK);
    }
```

Get Single book;

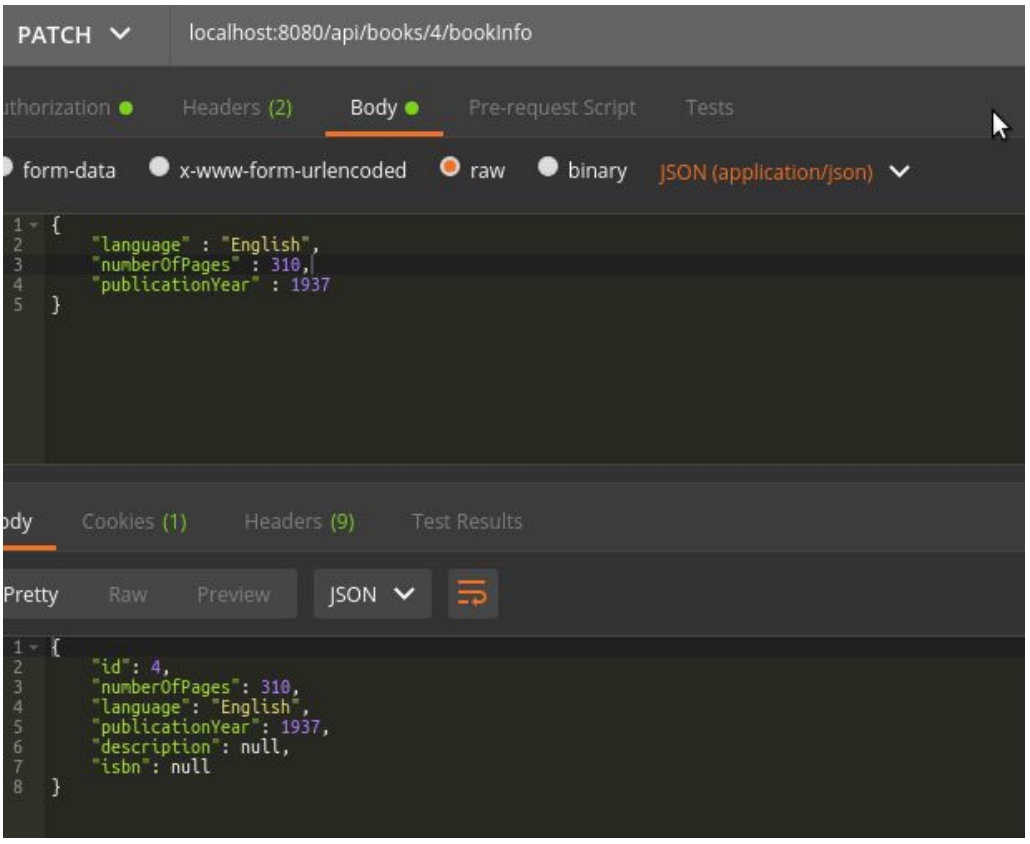
GET localhost:8080/api/books/4

Body Cookies (1) Headers (9) Test Results

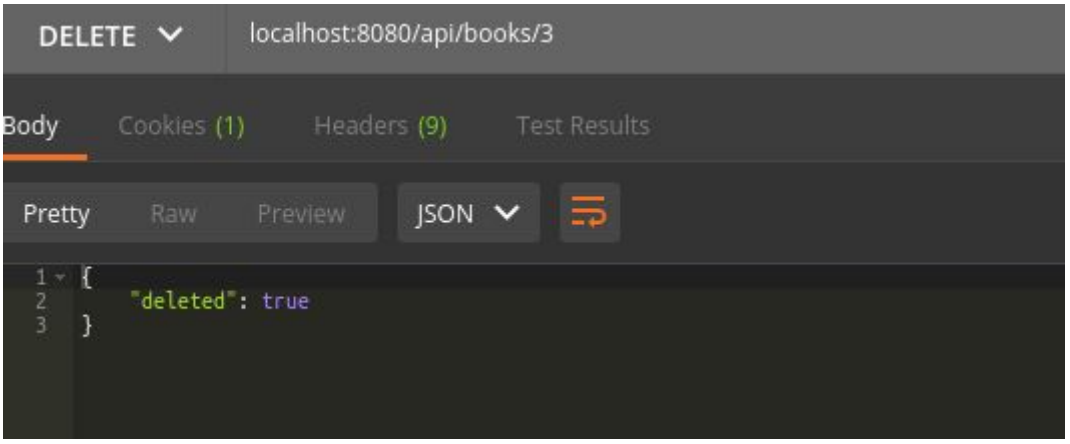
Pretty Raw Preview JSON


```
{
    "id": 4,
    "title": "The Hobbit",
    "authors": [
        {
            "id": 3,
            "name": "J. R. R. Tolkien",
            "description": "An English writer, poet, philologist, and university professor who is best known as the author of the classic h
                Silmarillion. While many other authors had published works of fantasy before Tolkien,[3] the great success of The Hobbit an
                of the genre. This has caused Tolkien to be popularly identified as the \"father\" of modern fantasy literature—or, more pr
                on a list of \"The 50 greatest British writers since 1945\". Forbes ranked him the 5th top-earning \"dead celebrity\" in 20
        }
    ],
    "genres": [
        {
            "id": 5,
            "name": "Fantasy",
            "description": "fiction in a unreal setting that often includes magic, magical creatures, or the supernatural"
        }
    ]
}
```

Update Book Info;



Delete Book;



Apply Entry;

POST localhost:8080/api/entries

```
{
    "borrowedBookId" : 1,
    "borrowerUsername" : "TestUser"
}
```

Body Cookies (1) Headers (9) Test Results

Pretty Raw Preview JSON


```
{
    "id": 1,
    "bookBorrowed": {
        "id": 1,
        "title": "The Shining",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
                "description": "An American author of horror, supernatural fiction, suspense, science fi
                    adapted into feature films, miniseries, television series, and comic books."
            }
        ],
        "genres": [
            {
                "id": 4,
                "name": "Horror",
                "description": "fiction in which events evoke a feeling of dread and sometimes fear in b
            }
        ]
    },
    "userBorrowing": {
        "id": 3,
        "username": "TestUser"
    },
    "dateStarted": "2018-08-12T17:51:09.185+0000",
    "dateFinished": null,
    "returned": false
}
```

Entry Status;

PATCH localhost:8080/api/entries/1

```
{
    "returned" : true
}
```

ody Cookies (1) Headers (9) Test Results

Pretty Raw Preview JSON


```
{
    "id": 1,
    "bookBorrowed": {
        "id": 1,
        "title": "The Shining",
        "authors": [
            {
                "id": 1,
                "name": "Stephen King",
                "description": "An American author of horror, supernatural fiction, suspense, science fiction and fa
                    adapted into feature films, miniseries, television series, and comic books."
            }
        ],
        "genres": [
            {
                "id": 4,
                "name": "Horror",
                "description": "fiction in which events evoke a feeling of dread and sometimes fear in both the char
            }
        ]
    },
    "userBorrowing": {
        "id": 3,
        "username": "TestUser"
    },
    "dateStarted": "2018-08-12T17:51:09.000+0000",
    "dateFinished": "2018-08-12T17:54:48.895+0000",
    "returned": true
}
```


Filter Entry by Status;

GET localhost:8080/api/entries?returned=true

Pretty Raw Preview JSON


```
[
    {
        "id": 1,
        "bookBorrowed": {
            "id": 1,
            "title": "The Shining",
            "authors": [
                {
                    "id": 1,
                    "name": "Stephen King",
                    "description": "An American author of horror, supernatural fiction, suspense, science fiction
                        adapted into feature films, miniseries, television series, and comic books."
                }
            ],
            "genres": [
                {
                    "id": 4,
                    "name": "Horror",
                    "description": "fiction in which events evoke a feeling of dread and sometimes fear in both t
                }
            ]
        },
        "userBorrowing": {
            "id": 3,
            "username": "TestUser"
        },
        "dateStarted": "2018-08-12T17:51:09.000+0000",
        "dateFinished": "2018-08-12T17:54:48.000+0000",
        "returned": true
    }
]
```

CHAPTER FIVE

CONCLUSION AND FUTURE WORKS

Project functions are finished and running succesful. UI designs will be finished in the next phase. In the project, react will be used for frontend instead of swing.