CS6.401 Software Engineering
Spring 2024
Project - 2

Team - 22

Gunank Singh Jakhar (2022201057)

M Elamparithy (2022801014)

Rishabh Gupta (2022202011)

Piyush Singh (2022201032)

Prashant Kumar (2022201058)

Feature 1: Better user management

Limitations: Only the admin can create a new account for a user. In addition, only the admin can edit and update user credentials.

Improvement: Enabled user self-registration.

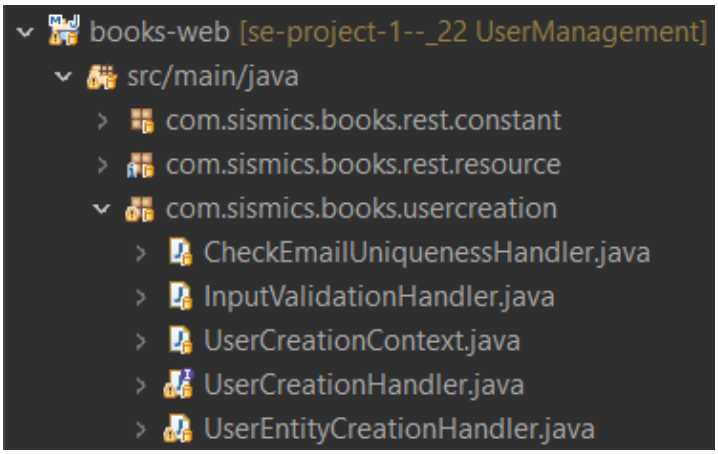
Design Pattern: Chain of Responsibility

The Chain of Responsibility is a behavioral design pattern that allows one to pass requests along a chain of handlers. Each handler processes the request or passes it on to the next handler in the chain. This design pattern decouples the sender of a request from its receivers, allowing multiple objects to handle the request. Each handler has a reference to the next handler in the chain.

The steps followed to register a user are:

1. Validation of Input (username and email uniqueness, password confirmation)
2. Creation of a User object with the inputs as the attributes
3. Saving the object in the database

Directory Structure



File Description

UserCreationContext.java *(Has all the input attributes, its object is passed as input to all the concrete handlers)*

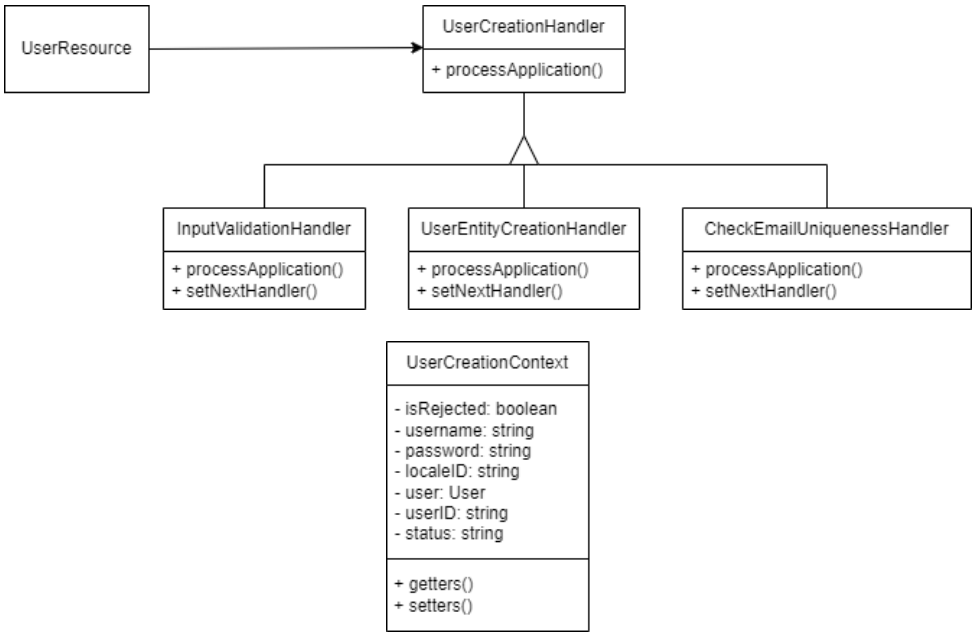
UserCreationHandler.java *(Handler Interface)*

InputValidationHandler.java *(validates username, email, and password)*

UserEntityCreationHandler.java *(creates user object and populates its attributes with the validated inputs)*

CheckEmailUniquenessHandler.java *(Checks if input email belongs to a registered user or not)*

UML Diagrams



Code Snippets

createAccount() function in **UserResource** class.

UserResource.java

```java
    @PUT
    @Path("/create-account")
    public Response createAccount(
            @FormParam("username") String username,
            @FormParam("password") String password,
            @FormParam("locale") String localeId,
            @FormParam("email") String email) throws Exception {

        UserCreationContext userCreationContext = new UserCreationContext(username,password,localeId,email);

        //validate the input data
        InputValidationHandler inputValidationHandler = new InputValidationHandler();
        //create the user
        UserEntityCreationHandler userEntityCreationHandler = new UserEntityCreationHandler();
        //Raise a user creation event
        CheckEmailUniquenessHandler checkEmailUniquenessHandler = new CheckEmailUniquenessHandler();

        //setting up chain of responsibility
        inputValidationHandler.setNextHandler(userEntityCreationHandler);
        userEntityCreationHandler.setNextHandler(checkEmailUniquenessHandler);


        inputValidationHandler.processApplication(userCreationContext);

```

UserCreationContext class

UserCreationContext.java

```java
package com.sismics.books.usercreation;


import com.sismics.books.core.model.jpa.User;

public class UserCreationContext {
    private boolean isRejected;
    private String username;
    private String password;
    private String localeId;
    private String email;
    private User user;
    private String userId;
    private String status;

    public UserCreationContext(String username, String password, String localeId, String email) {
        isRejected = false;
        this.username = username;
        this.password = password;
        this.localeId = localeId;
        this.email = email;
        this.user = new User();
        this.setStatus("ok");
    }
```

UserCreationHandler class

UserCreationHandler.java

```java
package com.sismics.books.usercreation;

import org.codehaus.jettison.json.JSONException;

public interface UserCreationHandler {
    void processApplication(UserCreationContext userContext) throws Exception;
}
```

InputValidationHandler class

InputValidationHandler.java

```java
package com.sismics.books.usercreation;

import com.sismics.rest.util.ValidationUtil;

public class InputValidationHandler implements UserCreationHandler{

    private UserCreationHandler nextHandler;


    public void setNextHandler(UserCreationHandler nextHandler) {
        this.nextHandler = nextHandler;
    }
    @Override
    public void processApplication(UserCreationContext userContext) throws Exception {

        String username = userContext.getUsername();
        String password = userContext.getPassword();
        String localeId = userContext.getLocaleId();
        String email = userContext.getEmail();

        username = ValidationUtil.validateLength(username, "username", 3, 50);
        ValidationUtil.validateAlphanumeric(username, "username");
        password = ValidationUtil.validateLength(password, "password", 8, 50);
        email = ValidationUtil.validateLength(email, "email", 3, 50);
        ValidationUtil.validateEmail(email, "email");

        userContext.setUsername(username);
        userContext.setPassword(password);
        userContext.setLocaleId(localeId);
        userContext.setEmail(email);
```

UserEntityCreationHandler class

UserEntityCreationHandler.java

```java
package com.sismics.books.usercreation;

import java.util.Date;

public class UserEntityCreationHandler implements UserCreationHandler {

    private UserCreationHandler nextHandler;

    public void setNextHandler(UserCreationHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    @Override
    public void processApplication(UserCreationContext userContext) throws Exception {

        String username = userContext.getUsername();
        String password = userContext.getPassword();
        String email = userContext.getEmail();

        User user = userContext.getUser();

        user.setRoleId(Constants.DEFAULT_USER_ROLE);
        user.setUsername(username);
        user.setPassword(password);
        user.setEmail(email);
        user.setCreateDate(new Date());
        user.setLocaleId(Constants.DEFAULT_LOCALE_ID);

        if(nextHandler != null) {
            try {
                nextHandler.processApplication(userContext);
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
```

CheckEmailUniquenessHandler class

CheckEmailUniquenessHandler.java

```java
package com.sismics.books.usercreation;

import com.sismics.books.core.dao.jpa.UserDao;

public class CheckEmailUniquenessHandler implements UserCreationHandler{
    private UserCreationHandler nextHandler;


    public void setNextHandler(UserCreationHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    @Override
    public void processApplication(UserCreationContext userContext) throws Exception {
        User user = userContext.getUser();

        // Create the user
        UserDao userDao = new UserDao();

        // check for email uniqueness
        User u = userDao.checkIfEmailExists(user.getEmail());
        if (u != null) {
            userContext.setStatus("A User account already exists with the specified email");
            throw new ServerException("A User account already exists with the specified email", null);
        }

        try {
            userDao.create(user);
        } catch (Exception e) {
            if ("AlreadyExistingUsername".equals(e.getMessage())) {
                userContext.setStatus("A User account already exists with the specfied username");
                throw new ServerException("AlreadyExistingUsername", "Login already used", e);
            } else {
                throw new ServerException("UnknownError", "Unknown Server Error", e);
            }
```

Demo

Register

Username

Elamparithy

E-mail

m.elamparithy@research.iiit.ac.in

Password

•••••••

Too short

Password (confirm)

Password (confirm)

Password and password confirmation must match

Register

Register

Username

Elamparithy

E-mail

m.elamparithy@research.iiit.ac.in

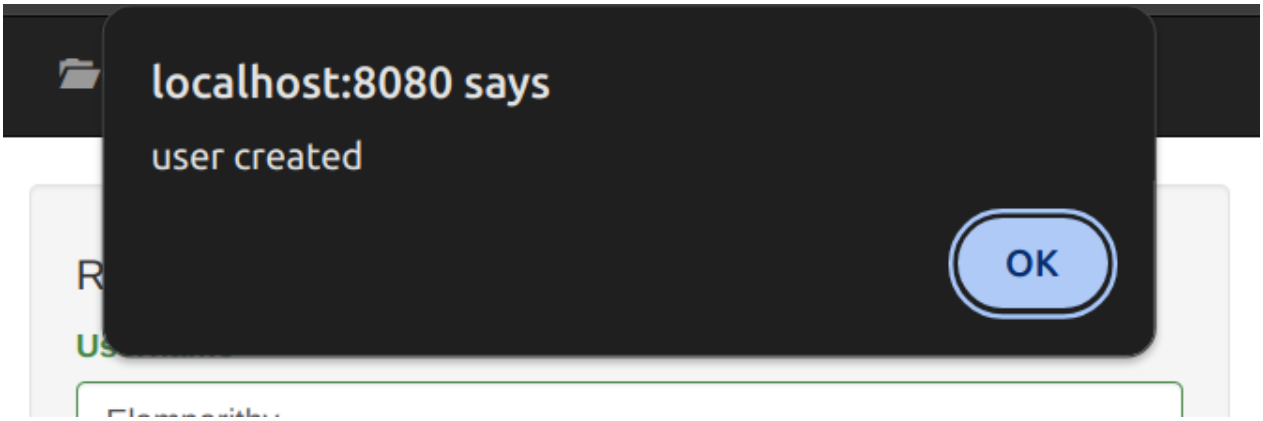
Password

••••••••

Password (confirm)

••••••••

Register



Login

Username

elamparithy

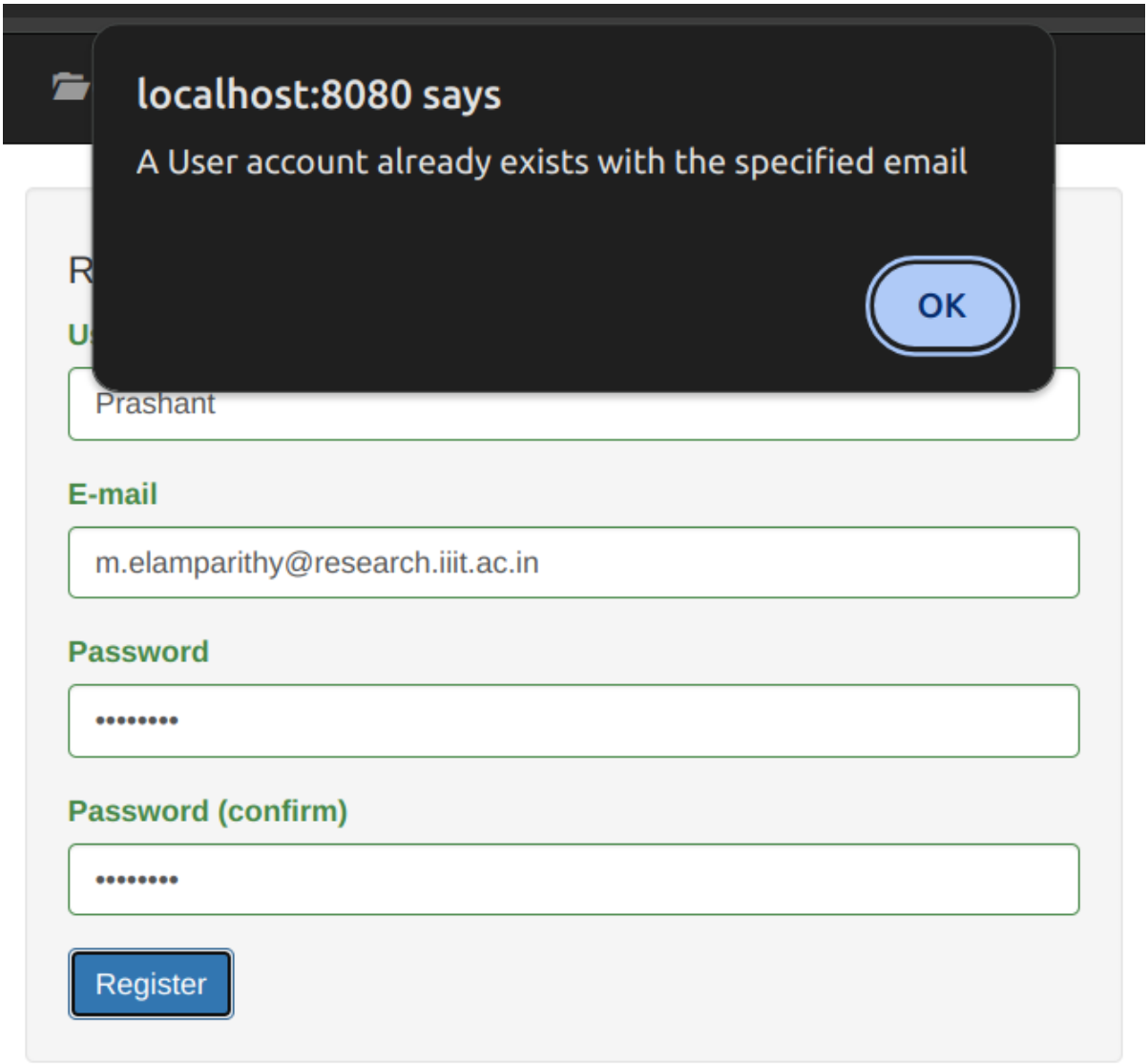
Password

••••••••

Remember me

Sign in

Not Registered? Create an Account



Feature 2: Common Library

To implement the common library feature, we have created a **CommonLibraryResource.java** file which has all the necessary functions to handle requests coming from the frontend.

CommonLibraryResource:

CommonLibraryResource.java

```
1  package com.sismics.books.rest.resource;
2
3  import javax.ws.rs.Path;
43
44 @Path("commonlibrary")
45 public class CommonLibraryResource extends BaseResource {
46
47     private void authenticateOrThrowForbidden() throws JSONException {
52
57     public Response addBookInLibrary(
110
114    public Response getAll() throws JSONException {
135
139    public Response cover(
162
166    public Response updateRating(
205
209    public Response getById(
231
235    public Response findBookByAuthor(
266
270    public Response getAllAuthorName()throws JSONException {
289
293    public Response findBookByGenre(
323
327    public Response getAllGenreName()throws JSONException {
346
350    public Response findBookByRating(
375
379    public Response getData(@QueryParam("param") String param) throws JSONException {
418
419 }
420
```

Also, we made a separate table called **“T_COMMON_LIBRARY”** which signifies books added in the common library and class **CommonLibrary** for common library books to store all information and created **CommonLibraryDao** to perform **CRUD** operations.

CommonLibrary:

CommonLibrary.java

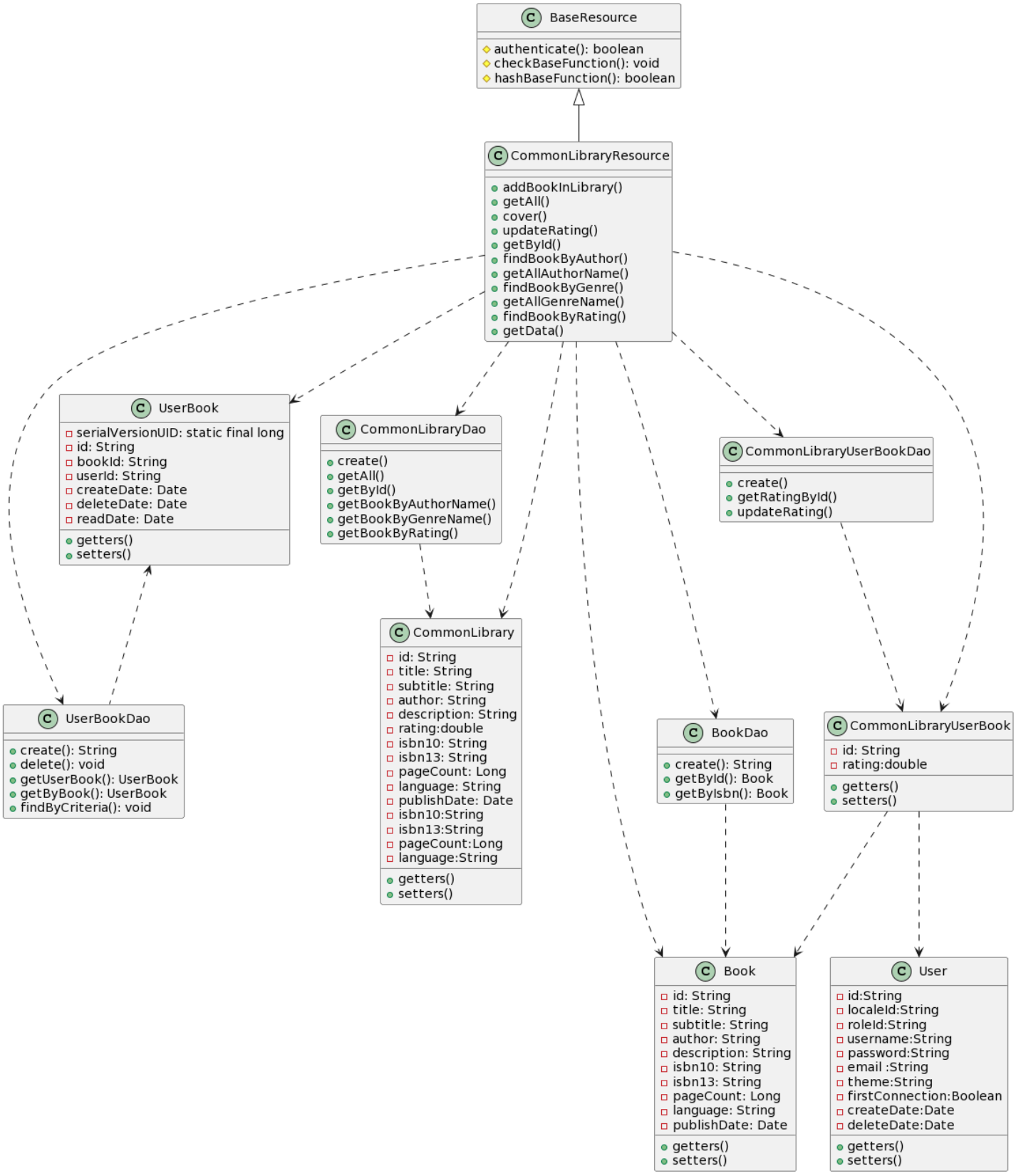
```java
package com.sismics.books.core.model.jpa;

import java.util.ArrayList;

@Entity
@Table(name = "T_COMMON_LIBRARY")
public class CommonLibrary {

    private String id;

    private String title;

    private String subtitle;

    private String author;

    private String description;

    private String genre;

    private double rating;

    private Integer noOfRatings;

    private String isbn10;

    private String isbn13;

    private Long pageCount;

    private String language;

    private Date publishDate;
```

CommonLibraryDao:

CommonLibraryDao.java

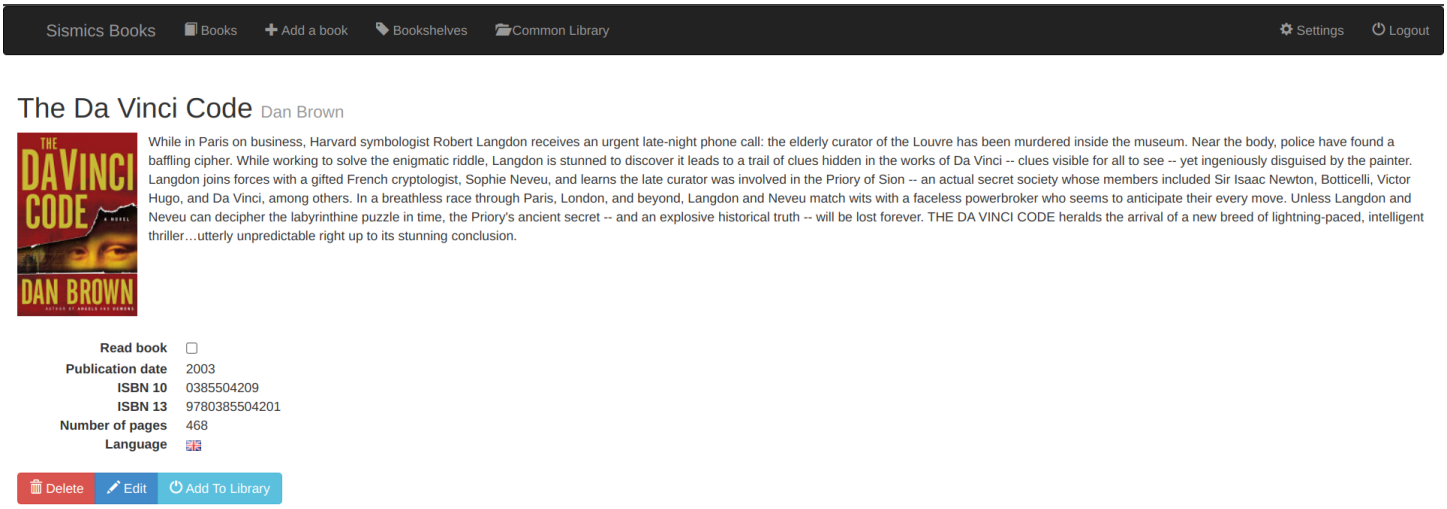
```java
package com.sismics.books.core.dao.jpa;

import java.util.List;

public class CommonLibraryDao {

    public String create(CommonLibrary commonLibrary) {

    public List<CommonLibrary> getAll(){

    public CommonLibrary getById(String id) {

    public List<CommonLibrary> getBookByAuthorName(String authorName) {

    public List<CommonLibrary> getBookByGenreName(String genreName) {

    public List<CommonLibrary> getBookByRating(double rating) {

}
```

UML Diagram of Feature 2 Implementation:

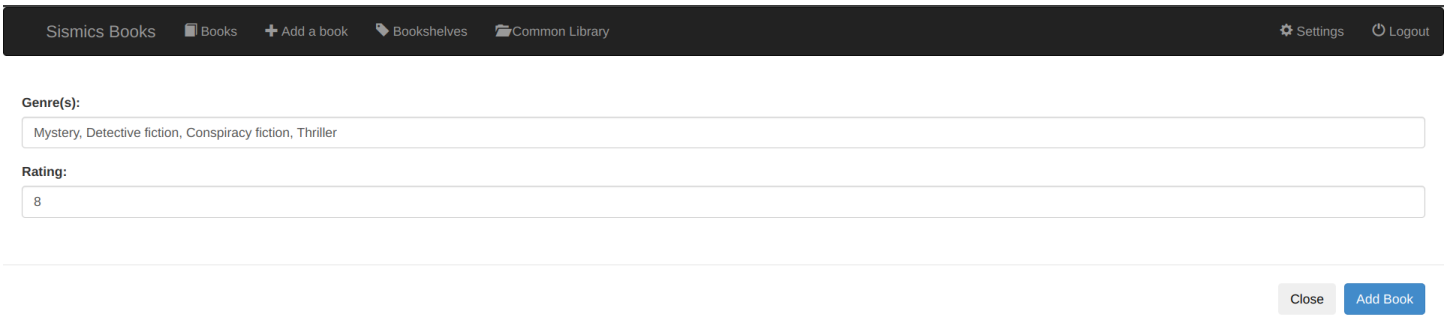


Demo:

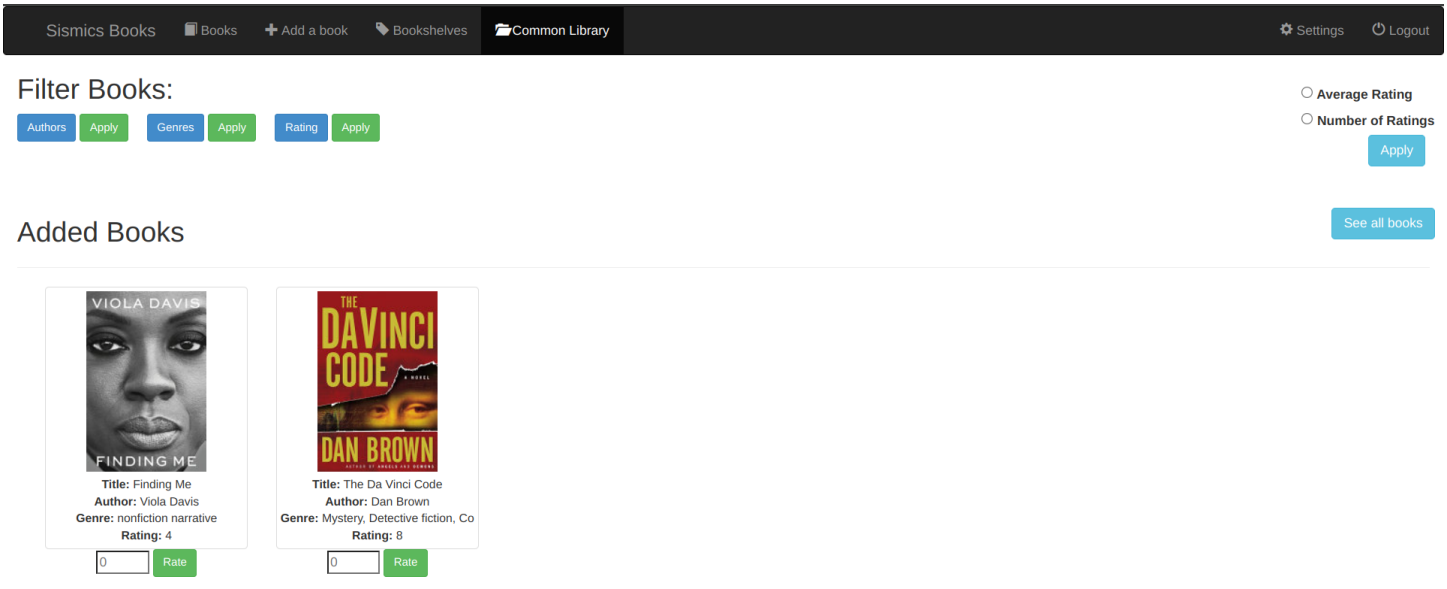
We added a button on each book **“Add To Library”** to add the particular book to the common library for public accessibility.



When we click on the **“Add To Library”** Button, we take two values from user **Genres** and **Rating,** and then only add them to the common library area.

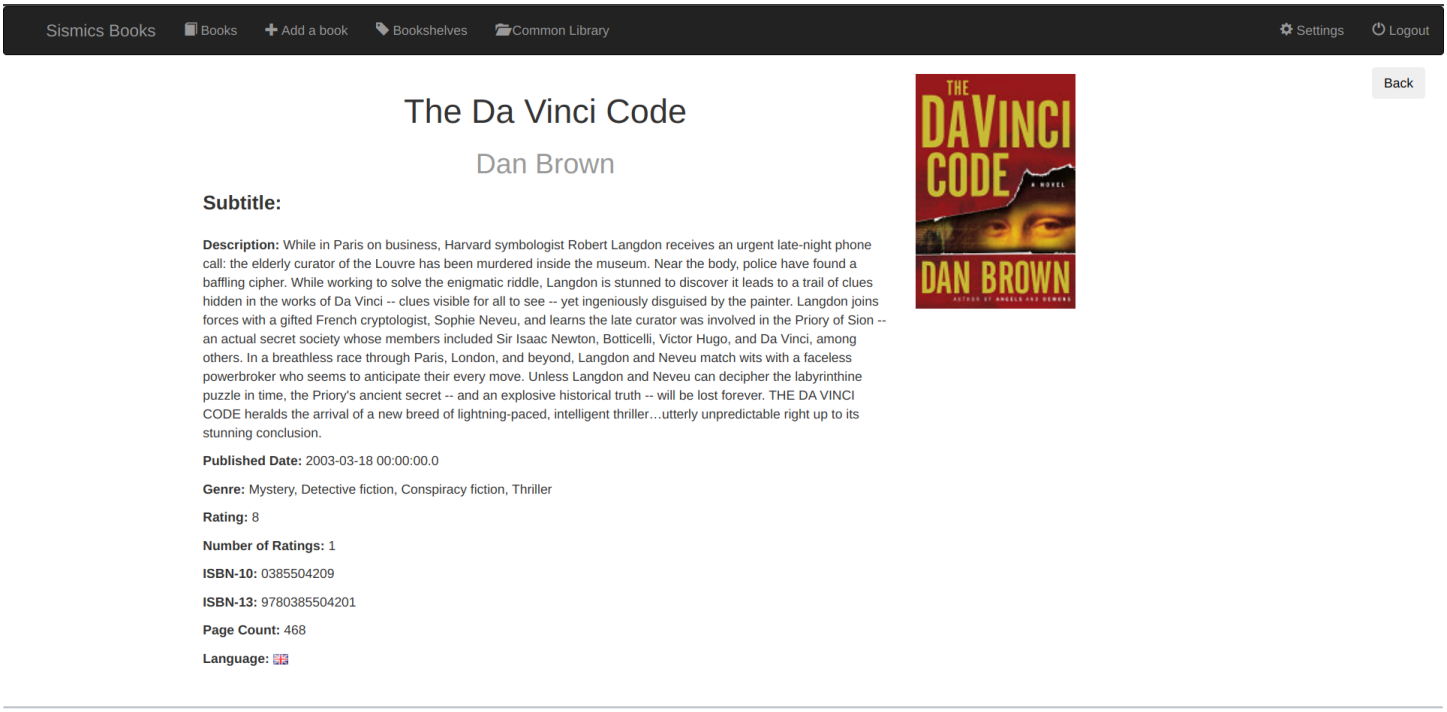


Here all authenticated users can see all books present in the common library.



Sub-feature: 1 Book Details

This is the view of books containing all the information like title, author, description, rating, ISBN, etc.



Sub-feature: 2 User Management

To resolve the issue of giving ratings multiple times to the same books in the common library, We created a new table **“T_COMMON_LIBRARY_USERBOOK”** which is used to store userId and bookId as primary key and rating as another attribute. The use of this table to check whether the user has given a rating or not.

CommonlibraryUserBook:

CommonlibraryUserBook.java

```java
package com.sismics.books.core.model.jpa;

import javax.persistence.Column;

@Entity
@Table(name = "T_COMMON_LIBRARY_USERBOOK")
public class CommonlibraryUserBook {
    @Id
    @Column(name = "CL_USERBOOK_ID", length = 255)
    private String id;

    @Column(name = "CL_USERBOOK_RATING")
    private double rating;

```

CommonLibraryDao to perform **CRUD** operations on the database.

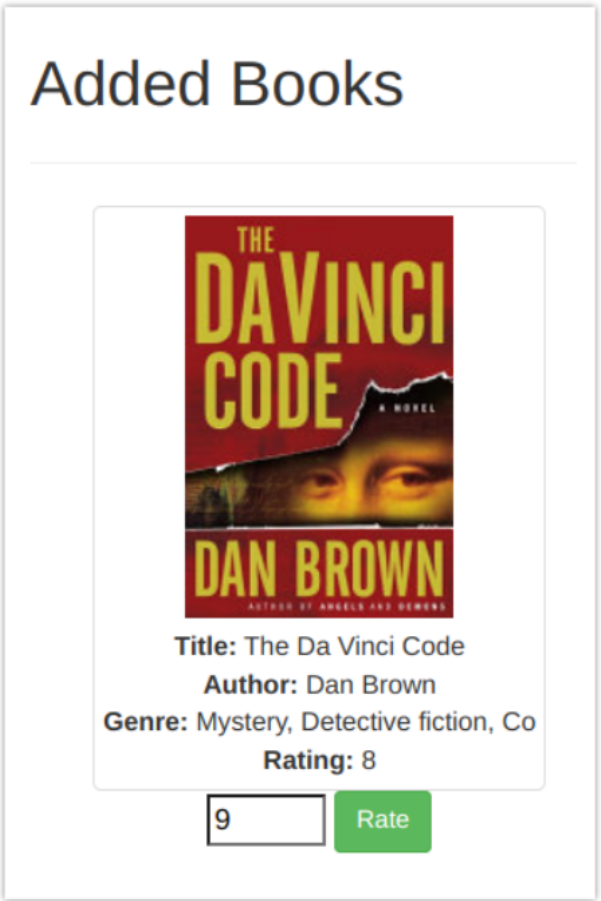
CommonlibraryUserBookDao.java

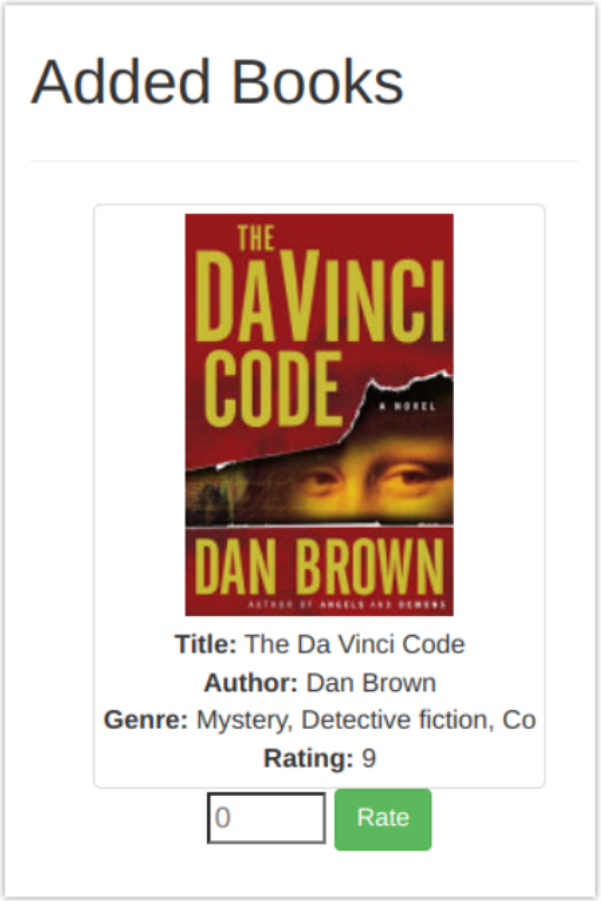
```java
package com.sismics.books.core.dao.jpa;

import javax.persistence.EntityManager;

public class CommonlibraryUserBookDao {

    public String create(CommonlibraryUserBook commonlibraryUserBook) {

    public CommonlibraryUserBook getRatingById(String userBookKey) {

    public void updateRating(CommonlibraryUserBook commonlibraryUserBook) {

}
```

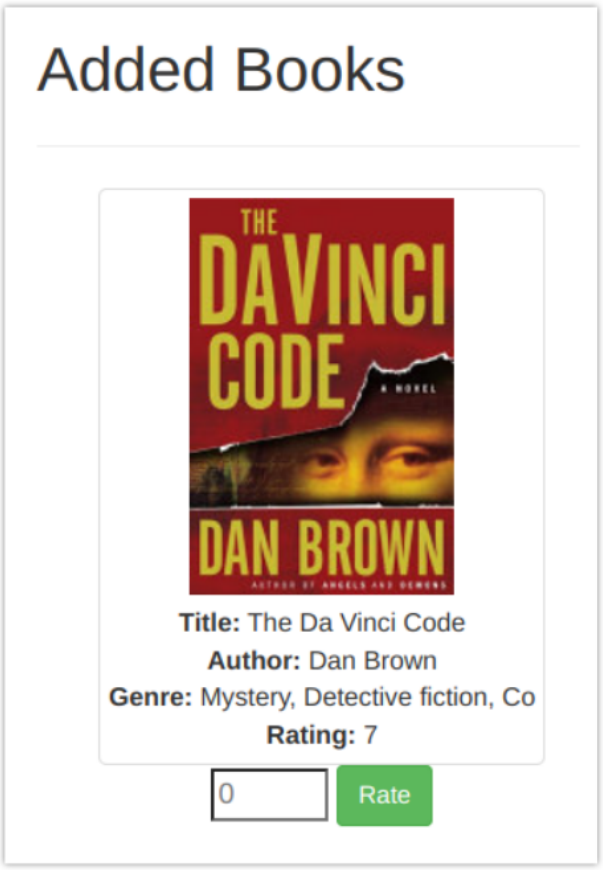
Any user can rate any book present in the common library.

But If the same user gives a rating to the same book, then it will be modified.

otherwise, the rating will be changed accordingly. (we are showing the average rating here)







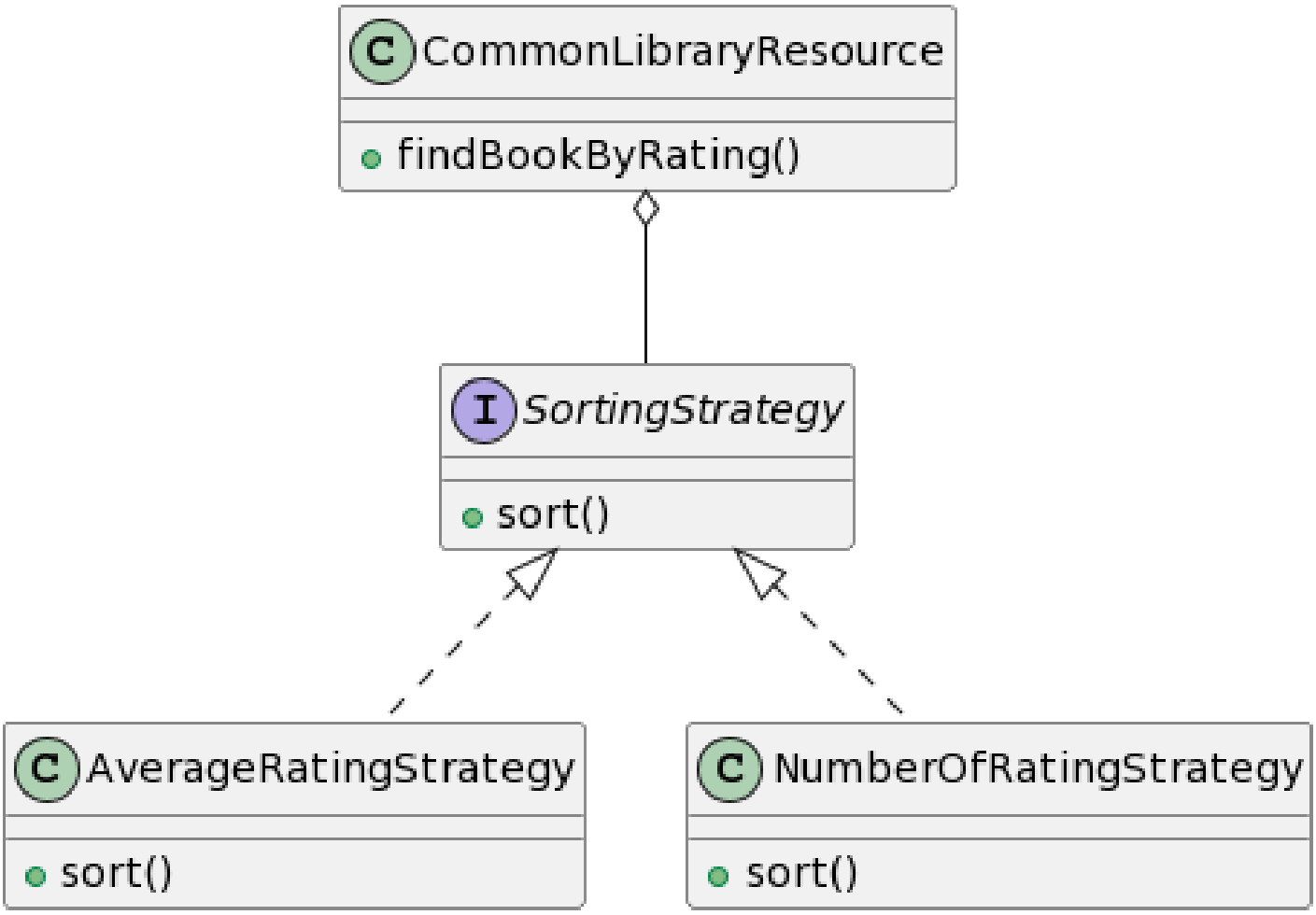
Sub-feature: 3 Book Ranking

Design Pattern: Strategy Design Pattern

The Strategy pattern is a behavioral design pattern that enables you to define a family of algorithms, encapsulate each one as an object, and make them interchangeable. This pattern allows the algorithms to vary independently from the clients that use them, making it easy to add new algorithms or modify existing ones without changing the clients.

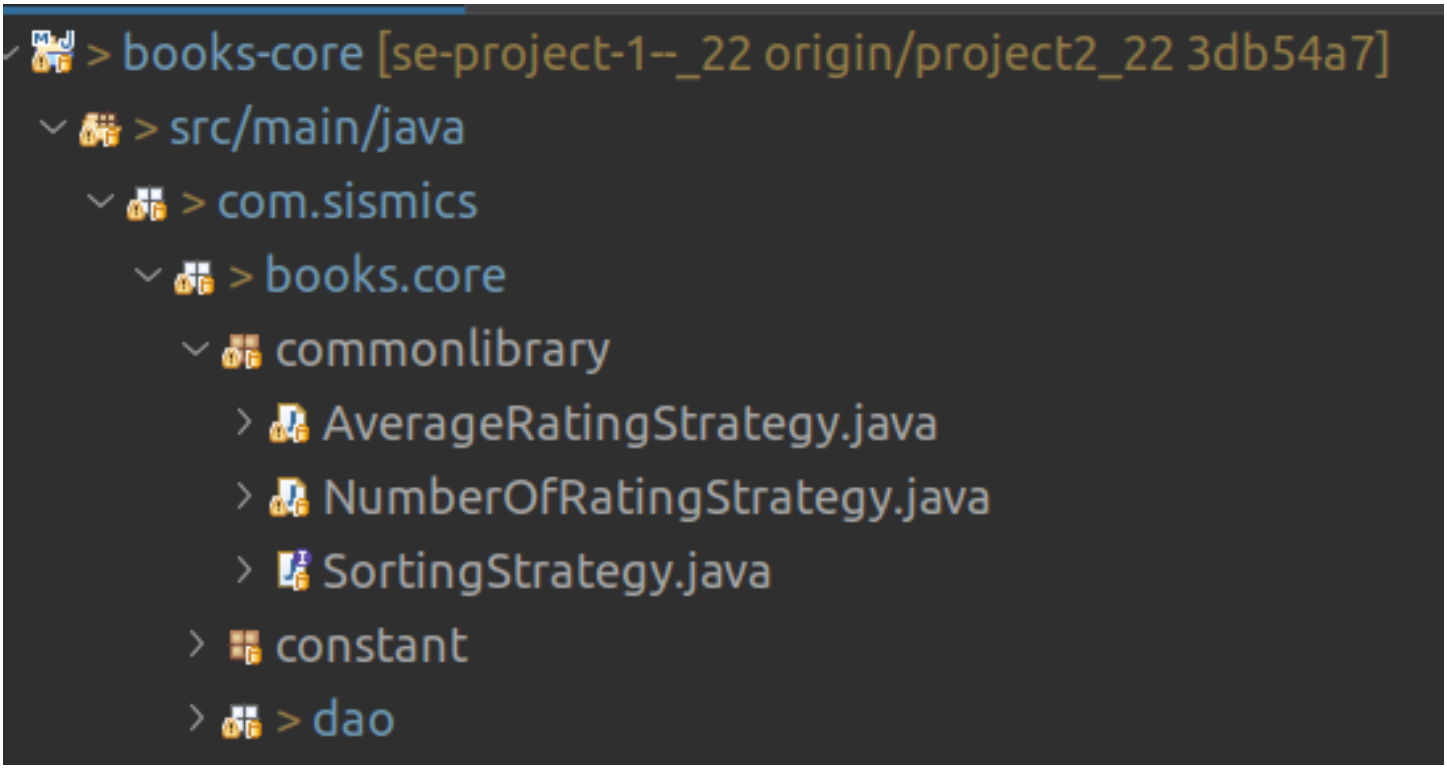
In this pattern, the strategy pattern defines a common interface for all the algorithms, allowing them to be used interchangeably. Each algorithm is encapsulated as a separate object that implements the common interface.

UML Diagram of Implementation:



CommonLibraryResource class serves the role of Context class in the Strategy pattern and uses the object of **SortingStrategy**. In this class, depending on the ranking condition (Sort by number of ratings or by average rating), the objects of **AverageRatingStrategy** and **NumberOfRatingStrategy** are assigned to the SortingStrategy object.

Directory Structure:



SortingStrategy - This abstract declares the sort method for the implementation.

AverageRatingStrategy - This class implements the sort method which sorts according to the average rating of each book.

NumberOfRatingStrategy - This class implements the sort method which sorts according to the number of ratings corresponding to a particular book.

SortingStrategy:

SortingStrategy.java

```
1  package com.sismics.books.core.commonlibrary;
2
3  import java.util.List;
6
7  public interface SortingStrategy {
8      public List<CommonLibrary>  sort(List<CommonLibrary> listAllBooks);
9  }
10
```

NumberOfRatingStrategy:

NumberOfRatingStrategy.java

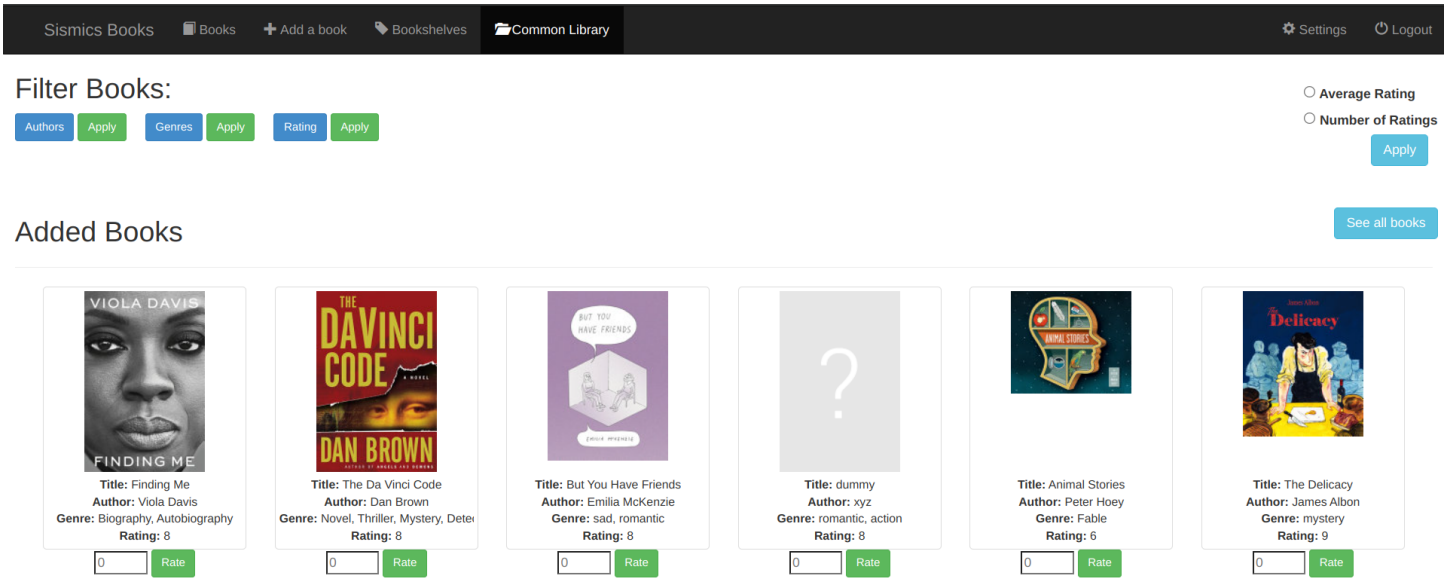
```java
package com.sismics.books.core.commonlibrary;

import java.util.ArrayList;

public class NumberOfRatingStrategy implements SortingStrategy {

    public List<CommonLibrary>  sort(List<CommonLibrary> books) {

        Collections.sort(books, new Comparator<CommonLibrary>() {
            @Override
            public int compare(CommonLibrary book1, CommonLibrary book2) {
                return -Double.compare(book1.getNoOfRatings(), book2.getNoOfRatings());
            }
        });
        return books;
    }
}

```

AverageRatingStrategy:

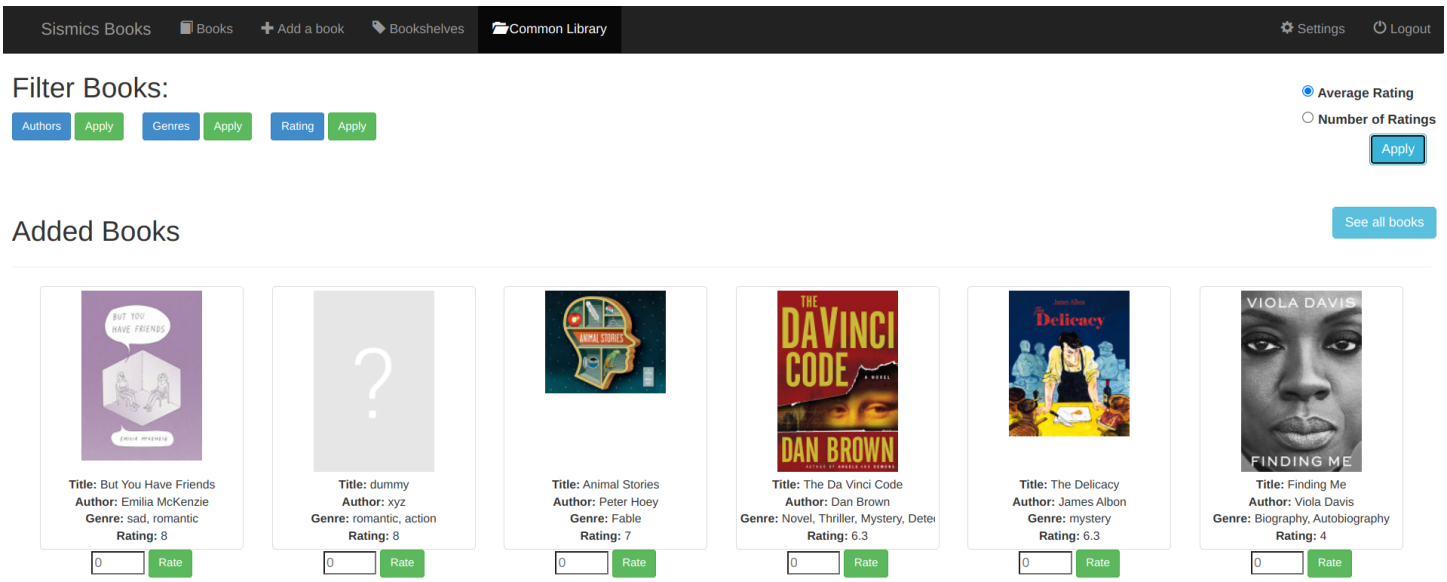
AverageRatingStrategy.java

```java

package com.sismics.books.core.commonlibrary;

import java.util.ArrayList;

public class AverageRatingStrategy implements SortingStrategy {
        public List<CommonLibrary>  sort(List<CommonLibrary> books) {

            Collections.sort(books, new Comparator<CommonLibrary>() {
                @Override
                public int compare(CommonLibrary book1, CommonLibrary book2) {
                    return -Double.compare(book1.getRating(), book2.getRating());
                }
            });
            return books;
    }
}

```

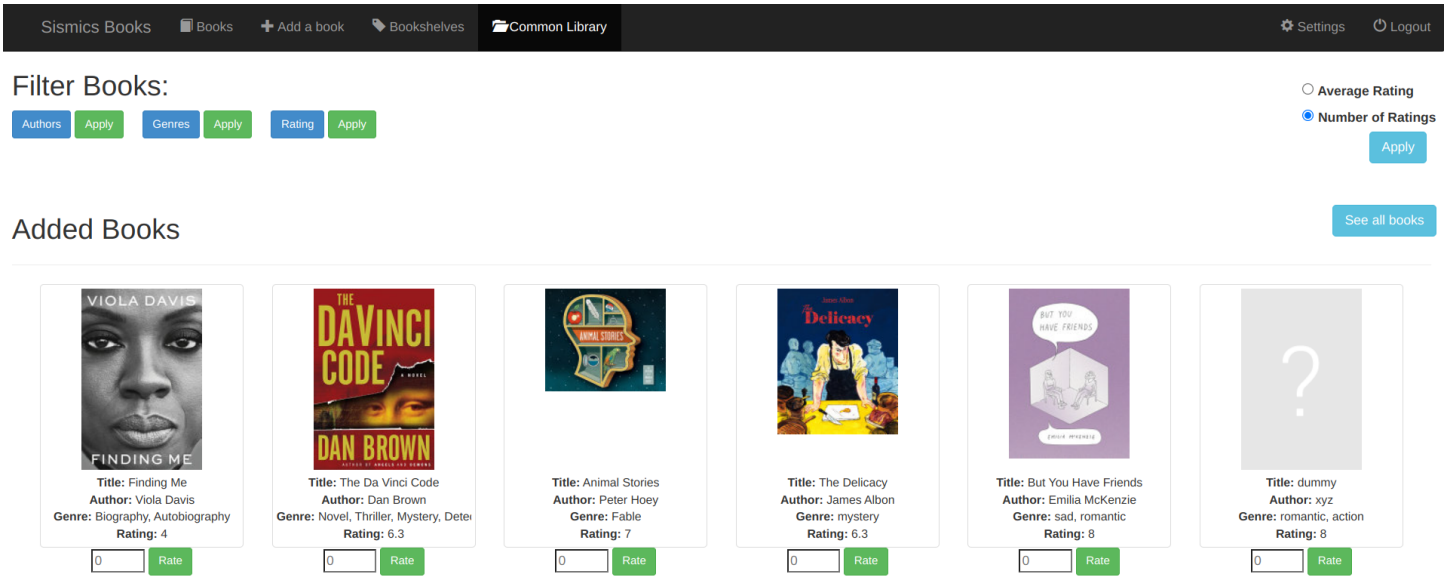
For the book ranking part, users can see the top 10 books based on average ratings or number of ratings.



Showing top books based on average ratings.



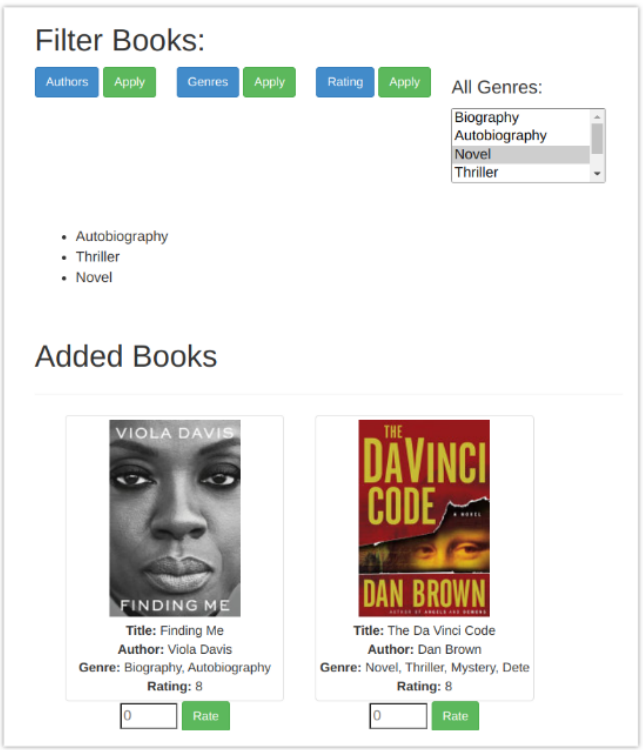
Showing top books based on the number of rating

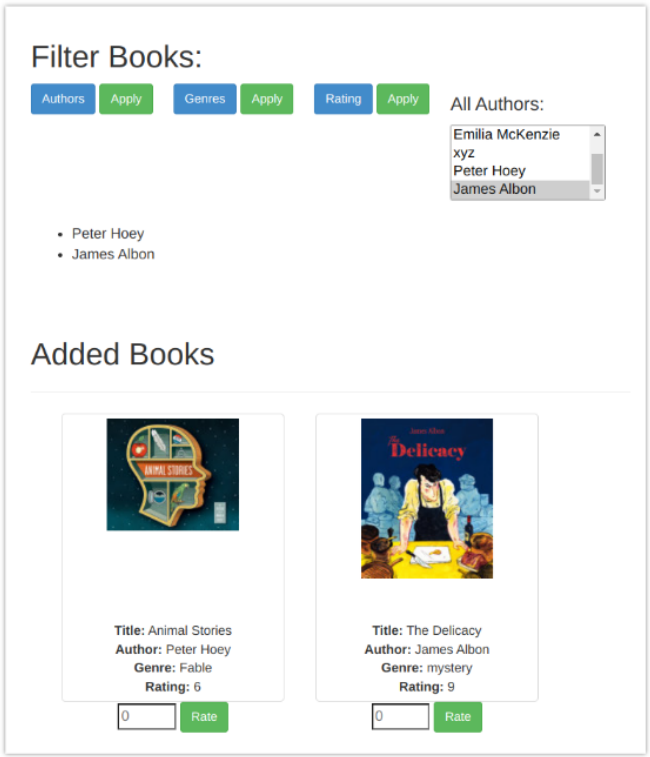


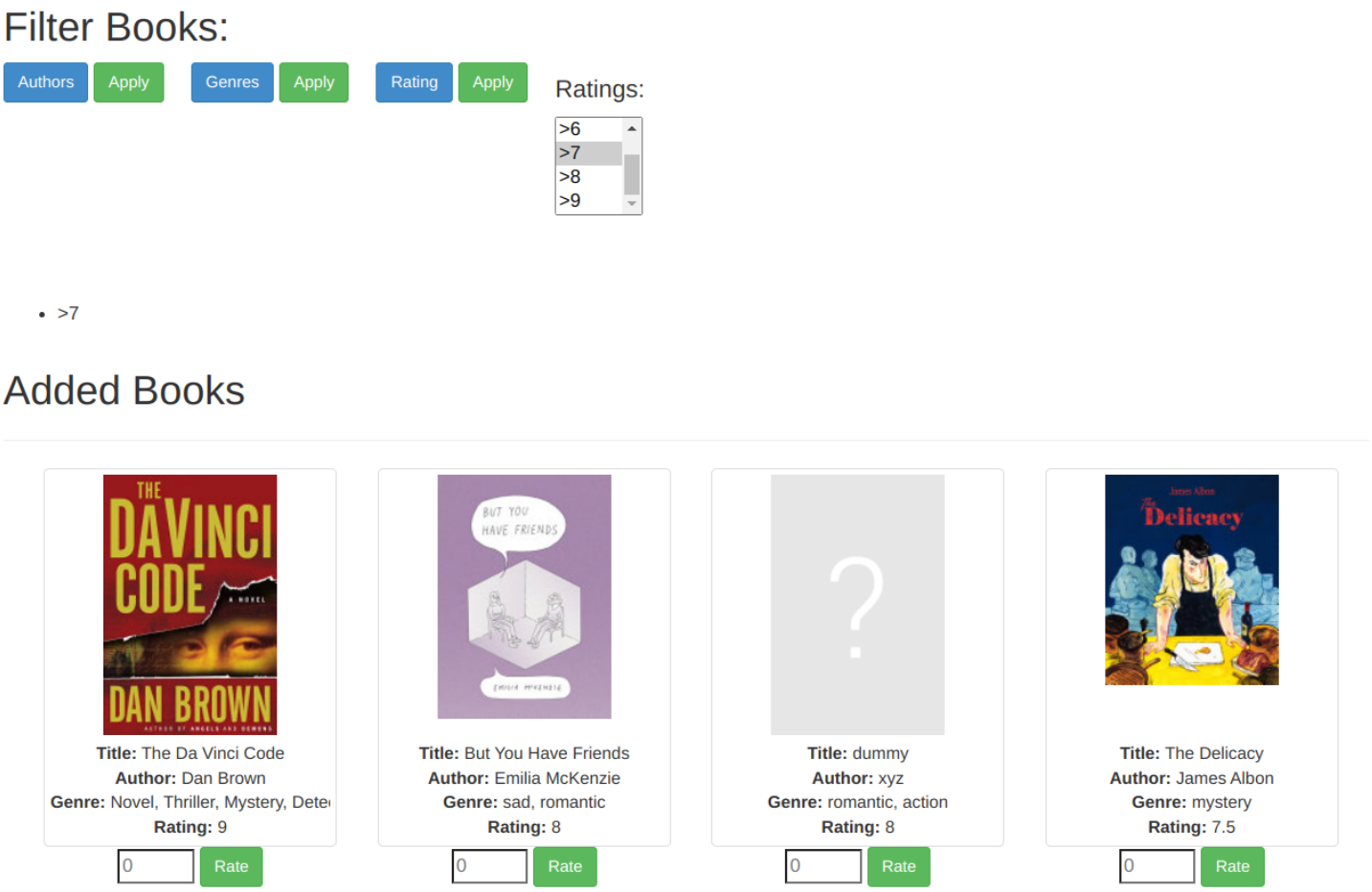
Sub-feature: 4 Filtering

Users can also filter out books based on multiple authors, multiple genres, and a single rating.

Based on the selected options we show only those books in the UI.

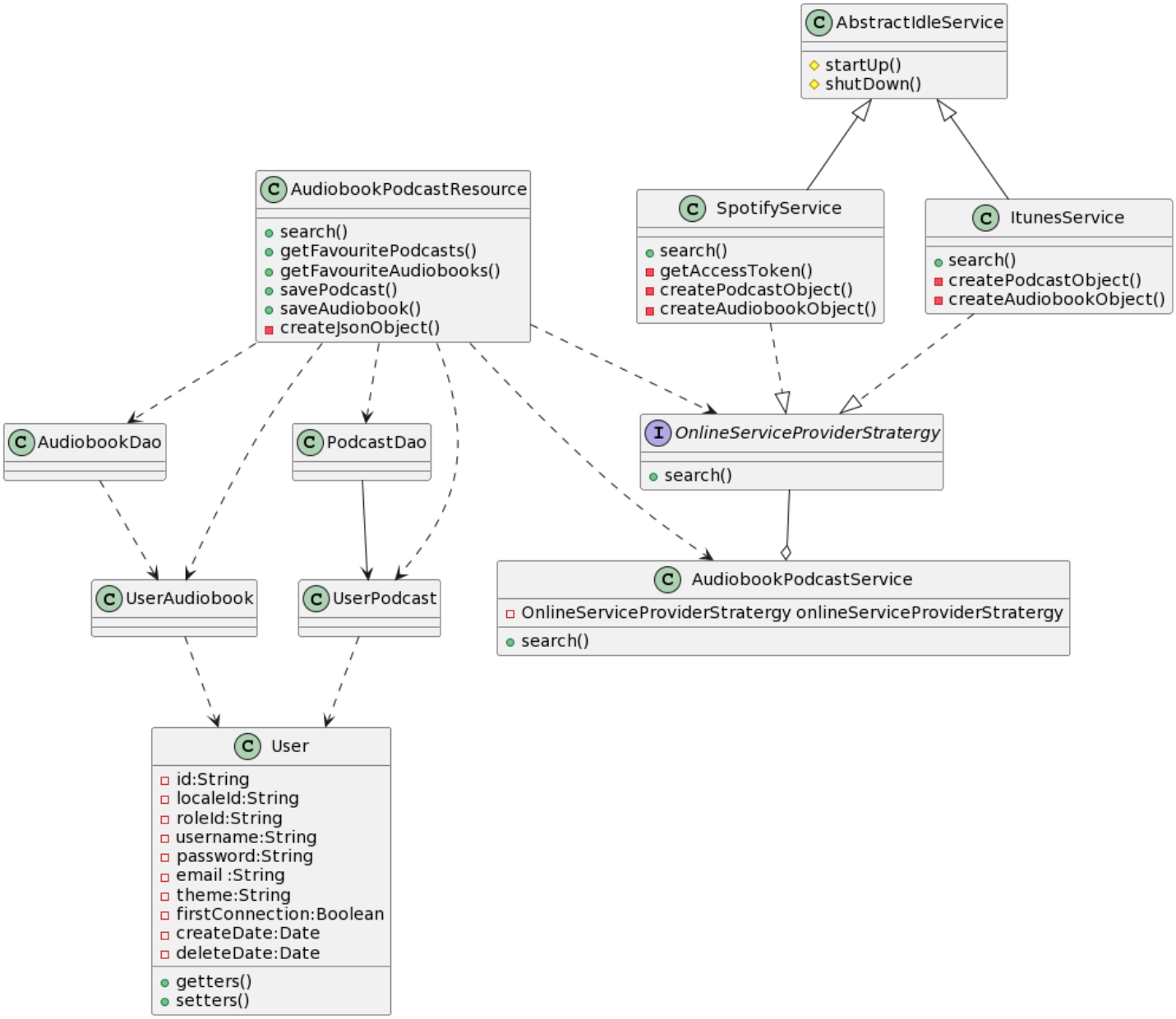






Feature 3: Online Integration

UML Diagram of Feature 3:

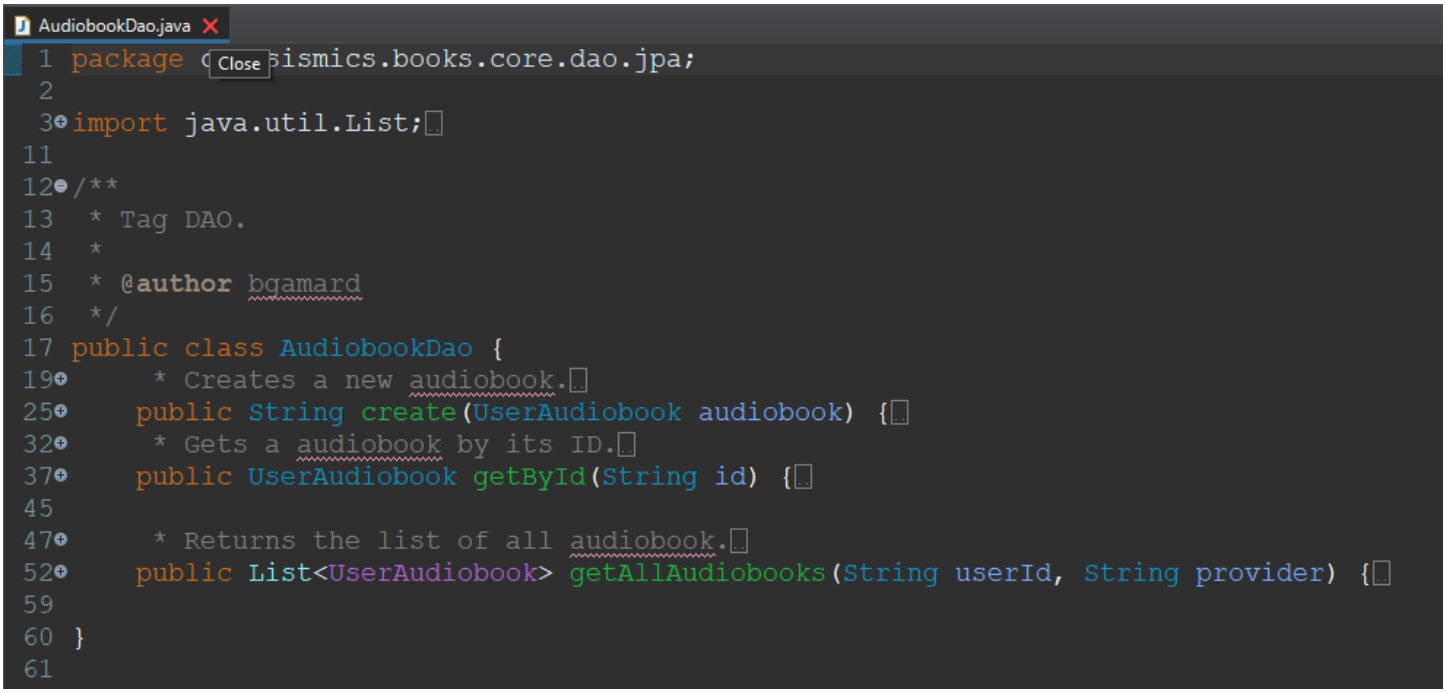


UserAudiobook: This class corresponds to the entity for the table (“T_AUDIOBOOK”) that stores all the audiobooks (whether obtained from Spotify or iTunes) favourited by a user.

UserAudiobook.java

```java
@Entity
@Table(name = "T_AUDIOBOOK")
public class UserAudiobook {
    /**
     * UserAudiobook ID.
     */
    private String id;

    /**
     * User ID.
     */
    private String userId;

    /**
     * UserAudiobook Author.
     */
    private String author;

    /**
     * UserAudiobook Preview Url.
     */
    private String previewUrl;

    /**
     * UserAudiobook Image Url.
     */
    private String imageUrl;

    /**
     * UserAudiobook Name
     */
    private String name;

    /**
     * UserAudiobook Description.
     */
    private String description;

    /**
     * UserAudiobook Narrator.
     */
    private String narrator;

    /**
     * UserPodcast Provider
     */
    private String provider;

```

AudiobookDao: This class is used to perform database queries on “T_AUDIOBOOK” table.



UserPodcast: This class corresponds to the entity for the table (“T_PODCAST”) that stores all the podcasts (whether obtained from Spotify or iTunes) favourited by a user.

UserPodcast.java

```java
package com.sismics.books.core.model.jpa;

import javax.persistence.Column;

@Entity
@Table(name = "T_PODCAST")
public class UserPodcast{
    /**
     * UserPodcast ID.
     */
    private String id;

    /**
     * User ID.
     */
    private String userId;

    /**
     * UserPodcast Audio Url.
     */
    private String audioUrl;

    /**
     * UserPodcast Image Url.
     */
    private String imageUrl;

    /**
     * UserPodcast Name
     */
    private String name;

    /**
     * UserPodcast Provider
     */
    private String provider;

```

PodcastDao: This class is used to perform database queries on “T_PODCAST” table.

PodcastDao.java

```java
package com.sismics.books.core.dao.jpa;

import java.util.List;

/**
 * Tag DAO.
 *
 * @author bgamard
 */
public class PodcastDao {
     * Creates a new podcast.
    public String create(UserPodcast podcast) {
     * Gets a podcast by its ID.
    public UserPodcast getById(String id) {

     * Returns the list of all podcasts.
    public List<UserPodcast> getAllPodcasts(String userId, String provider) {

}
```

AbstractIdleService: com.google.common.util.concurrent.AbstractIdleService class. Basically, throughout the codebase this class is used wherever we want a singleton class (e.g. some services). Thus, it made sense for concrete strategy classes to inherit from this class.

OnlineServiceProviderStrategy: corresponding to the Strategy interface class in the Strategy pattern.

OnlineServiceProviderStrategy.java

```java
package
books-core/src/main/java/com/sismics/books/core/service/OnlineServiceProviderStrategy.java

import java.io.IOException;

public interface OnlineServiceProviderStrategy{
    public ArrayNode search(String query, String type) throws IOException;
    public void saveToFavourites();
}
```

SpotifyService: corresponding to the concrete Strategy classes in the Strategy pattern.

SpotifyService.java

```
15 import org.codehaus.jackson.JsonNode;
16 import org.codehaus.jackson.map.ObjectMapper;
17 import org.codehaus.jackson.node.ArrayNode;
18 import org.codehaus.jackson.node.ObjectNode;
19 import org.slf4j.Logger;
20 import org.slf4j.LoggerFactory;
21
22 import com.google.common.util.concurrent.AbstractIdleService;
23
24
25 public class SpotifyService extends AbstractIdleService implements OnlineServiceProviderStrategy{
26
28      * Logger.
30     private static final Logger log = LoggerFactory.getLogger(SpotifyService.class);
31     final String CLIENT_ID="385a2ee78cc14c8793cf29c08aa63f91";
32     final String CLIENT_SECRET="f3e4f9a58d2d4fd6a4a45dad6cfd6be0";
33     final String AUTH_URL = "https://accounts.spotify.com/api/token";
34
36     protected void startUp() throws Exception {
41
43     protected void shutDown() throws Exception {
48
49     private String getAccessToken() throws IOException {
77
79     public ArrayNode search(String query, String type) throws IOException {
100
101    private ArrayNode createPodcastObject(ArrayNode items) {
115
116    private ArrayNode createAudiobookObject(ArrayNode items) {
132
```

ItunesService: corresponding to the concrete Strategy classes in the Strategy pattern.

ItunesService.java

```
1  package com.sismics.books.core.service;
2
3  import java.io.IOException;
20
21
22 public class ItunesService extends AbstractIdleService implements OnlineServiceProviderStrategy{
23
25      * Logger.
27     private static final Logger log = LoggerFactory.getLogger(ItunesService.class);
28
30     protected void startUp() throws Exception {
35
37     protected void shutDown() throws Exception {
42
44     public ArrayNode search(String query, String type) throws IOException {
63
64     private ArrayNode createPodcastObject(ArrayNode items) {
78
79     private ArrayNode createAudiobookObject(ArrayNode items) {
95
```

AudiobookPodcastService: corresponding to the Context class in Strategy pattern

ItunesService.java AudiobookPodcastService.java

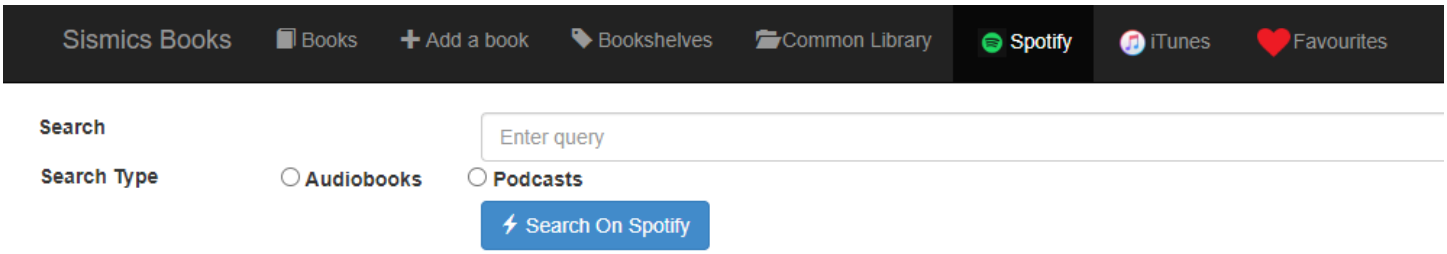
```java
package com.sismics.books.core.service;

import java.io.IOException;

public class AudiobookPodcastService {
    private OnlineServiceProviderStrategy onlineServiceProviderStrategy;

    public AudiobookPodcastService(OnlineServiceProviderStrategy onlineServiceProviderStrategy) {
        this.onlineServiceProviderStrategy = onlineServiceProviderStrategy;
    }
    public ArrayNode search(String query, String type) throws IOException {
        return this.onlineServiceProviderStrategy.search(query, type);
    }
}
```

AudiobookPodcastResource: This is a controller class which is the first endpoint where the frontend makes connection with backend.

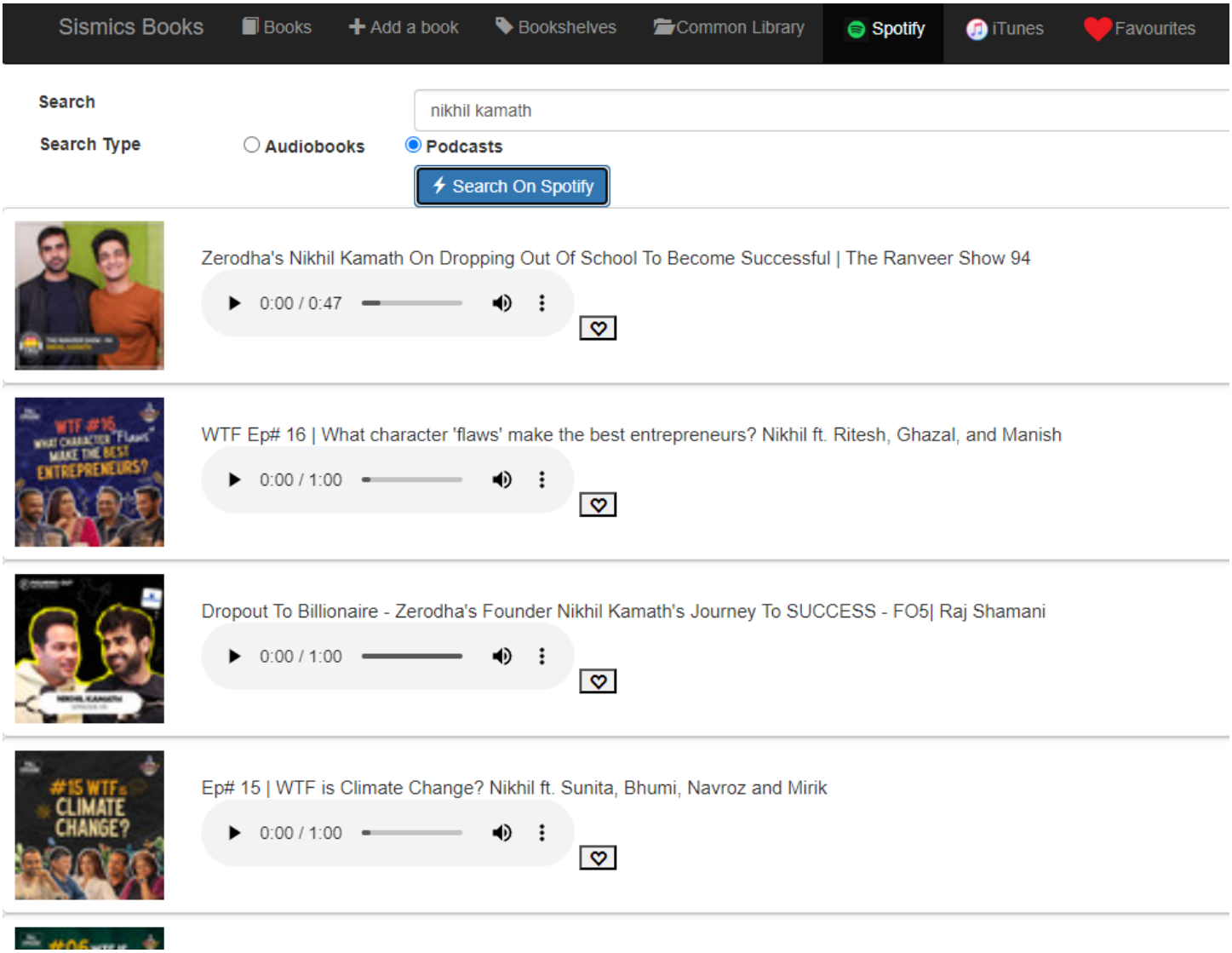
AudiobookPodcastResource.java

```java
package com.sismics.books.rest.resource;

import java.io.IOException;

/**
 * Tag REST resources.
 *
 * @author bgamard
 */
@Path("/thirdparty")
public class AudiobookPodcastResource extends BaseResource{
    public Response search(

    private List<JSONObject> createJsonObject(ArrayNode items) throws JSONException {

    public Response getFavouritePodcasts(

    public Response getFavouriteAudiobooks(

    public Response savePodcast(

    public Response saveAudiobook(
}
```

Demo:



Fetching podcasts from Spotify:



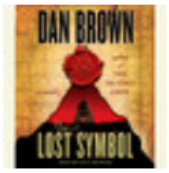
Fetching audiobook from Spotify:

Sismics Books | Books | Add a book | Bookshelves | Common Library | Spotify | iTunes | Favourites

Search dan brown

Search Type (•) **Audiobooks** ( ) **Podcasts**

Search On Spotify



The Lost Symbol
Author: Dan Brown
Narrator: Paul Michael
Author(s): Dan Brown
Narrator(s): Paul Michael
<p>In this stunning follow-up to the global phenomenon The Da V writer. The Lost Symbol is a masterstroke of storytelling-a deadly race through a real-world labyrinth of codes, secrets, chambers, tunnels, and temples of Washington, D.C., The Lost Symbol accelerates through a startling landscape towa unexpectedly to deliver an evening lecture in the U.S. Capitol Building. Within minutes of his arrival, however, the night takes a bizarre tu object as an ancient invitation . . . one meant to usher its recipient into a long-lost world of esoteric wisdom.

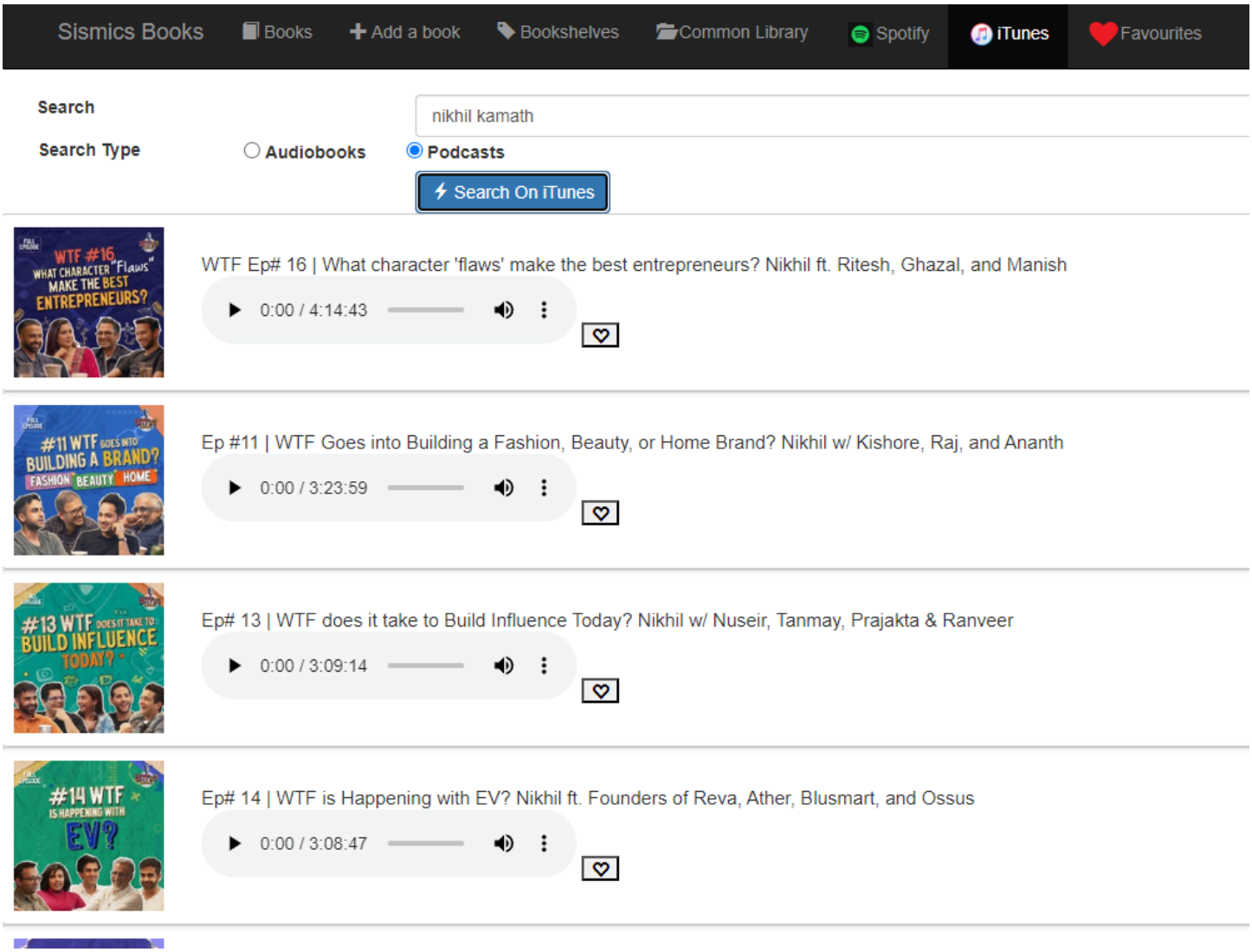
When Lan his only hope of saving Peter is to accept this mystical invitation and follow wherever it leads him. Langdon is instantly plunged into a cla toward a single, inconceivable truth.

As the world discovered in The Da Vinci Code and <b>Ang enigmatic codes. In this new novel, he again challenges readers with an intelligent, lightning-paced story that offers surprises at every tu yet.</p>

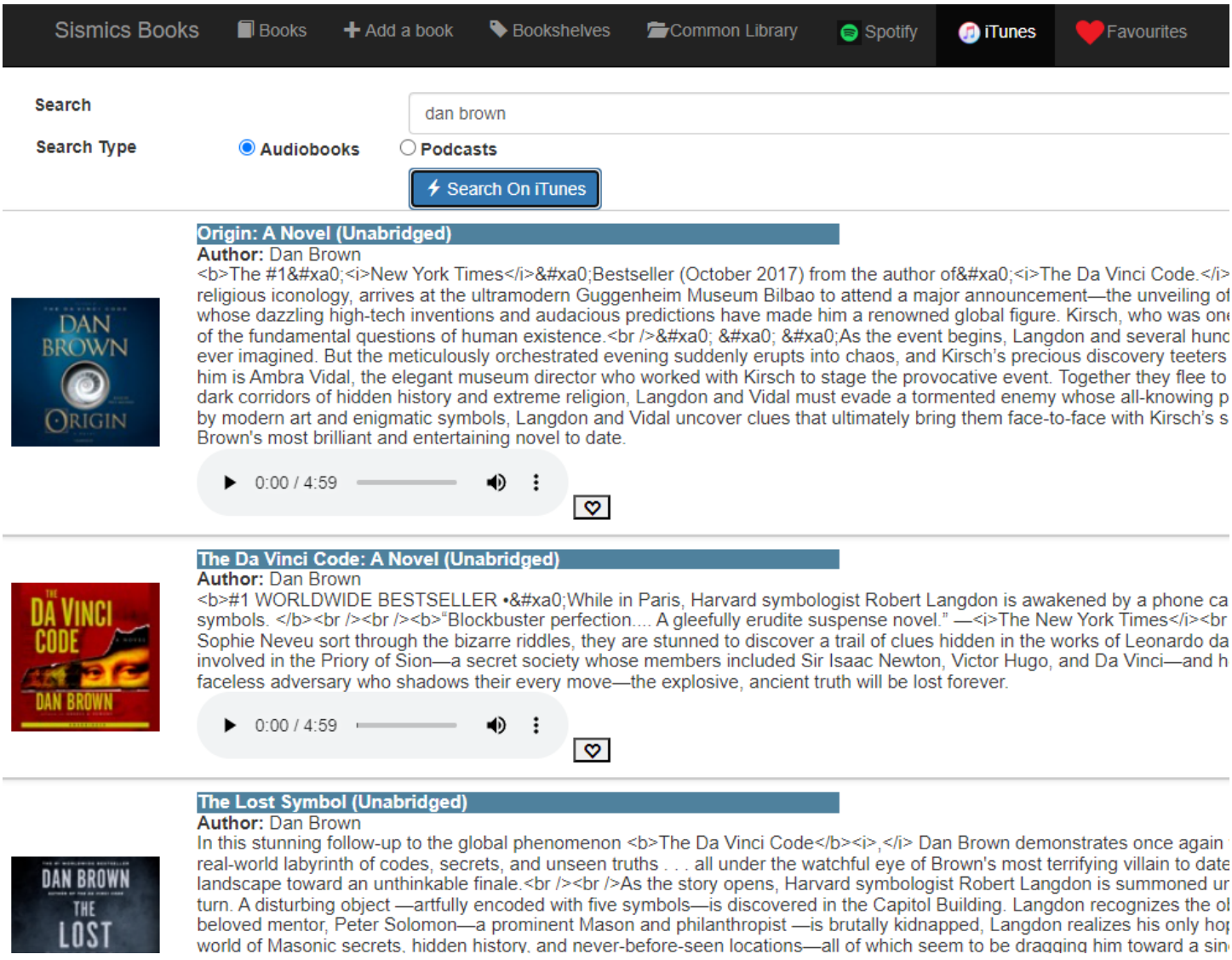
Take a Hint, Dani Brown: A Novel
Author: Talia Hibbert
Narrator: Ione Butler
Author(s): Talia Hibbert
Narrator(s): Ione Butler
<p>One of Oprah Magazine&#39 2020
</p><p>USA Today bestselling aut after a video of him "rescuing" her from their office building goes viral...
</p><p>Danika Brown knows w tension. But romance? Been there, done that, burned the T-shirt. Romantic partners, whatever their gender, are a distraction at best and score and knows their way around the bedroom. </p><p>When big, brooding security guard Zafir Ansari r destined to sleep together. But before she can explain that fact to him, a video of the heroic rescue goes viral. Suddenly, half the internet publicity. Lying to help children? Who on earth would refuse? </p><p>Dani's plan is simple: fake a relatio determined to corrupt Dani's stone-cold realism. Before long, he's tackling her fears into the dirt. But the former sports star has issues of now more complex than her thesis. Has her wish backfired? Is her focus being tested? Or is the universe just waiting for her to take a hi #1 New York Times bestselling author of the Little Bridge Island and Princess Diaries series </p&g

The Da Vinci Code: A Novel
Author: Dan Brown
Narrator: Paul Michael
Author(s): Dan Brown
Narrator(s): Paul Michael
While in Paris on business, Harvard symbologist Robert Langdon receives an have found a baffling cipher. While working to solve the enigmatic riddle, Langdon is stunned to discover it leads to a trail of clues hidden

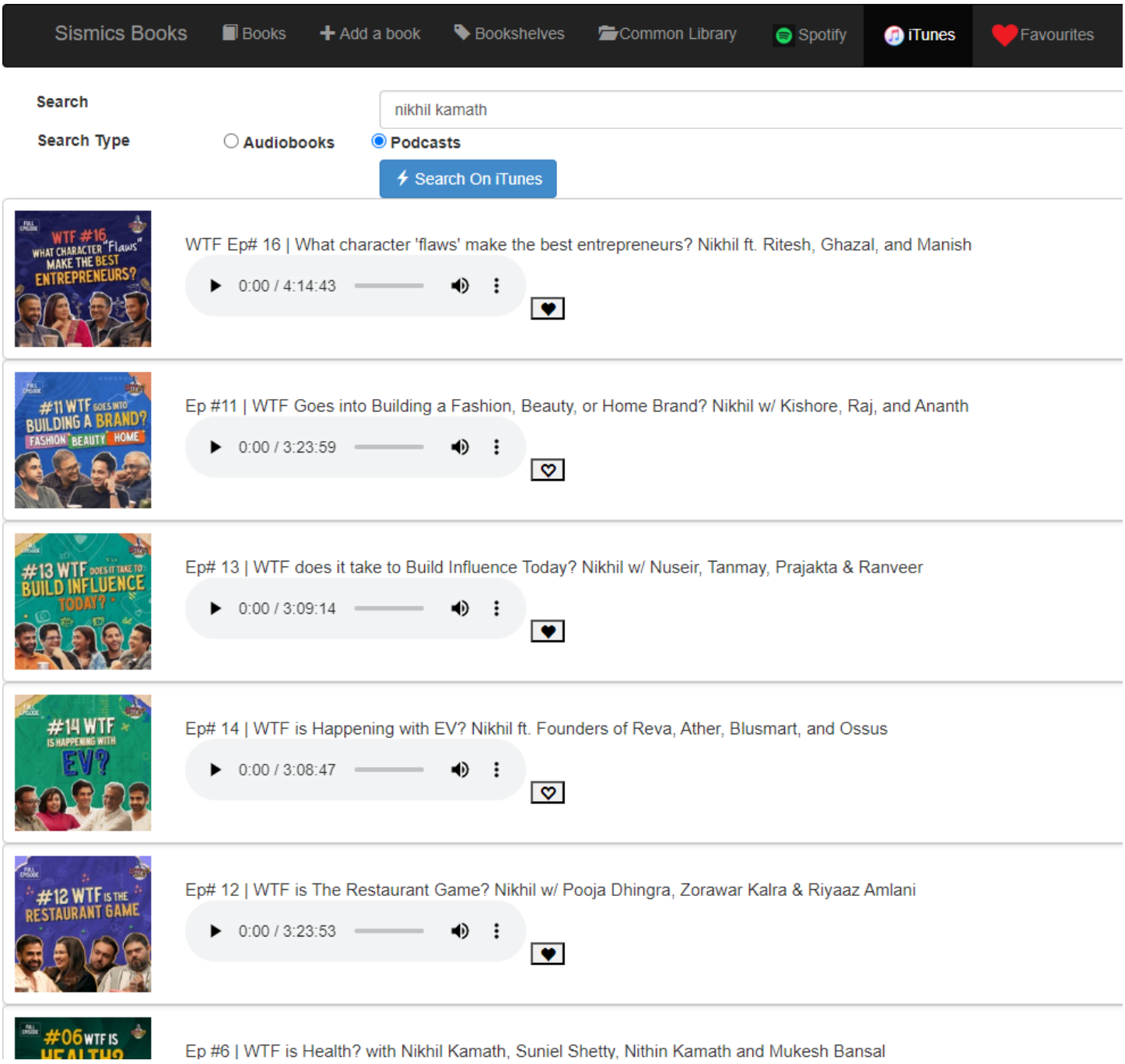
Fetching podcasts from iTunes:



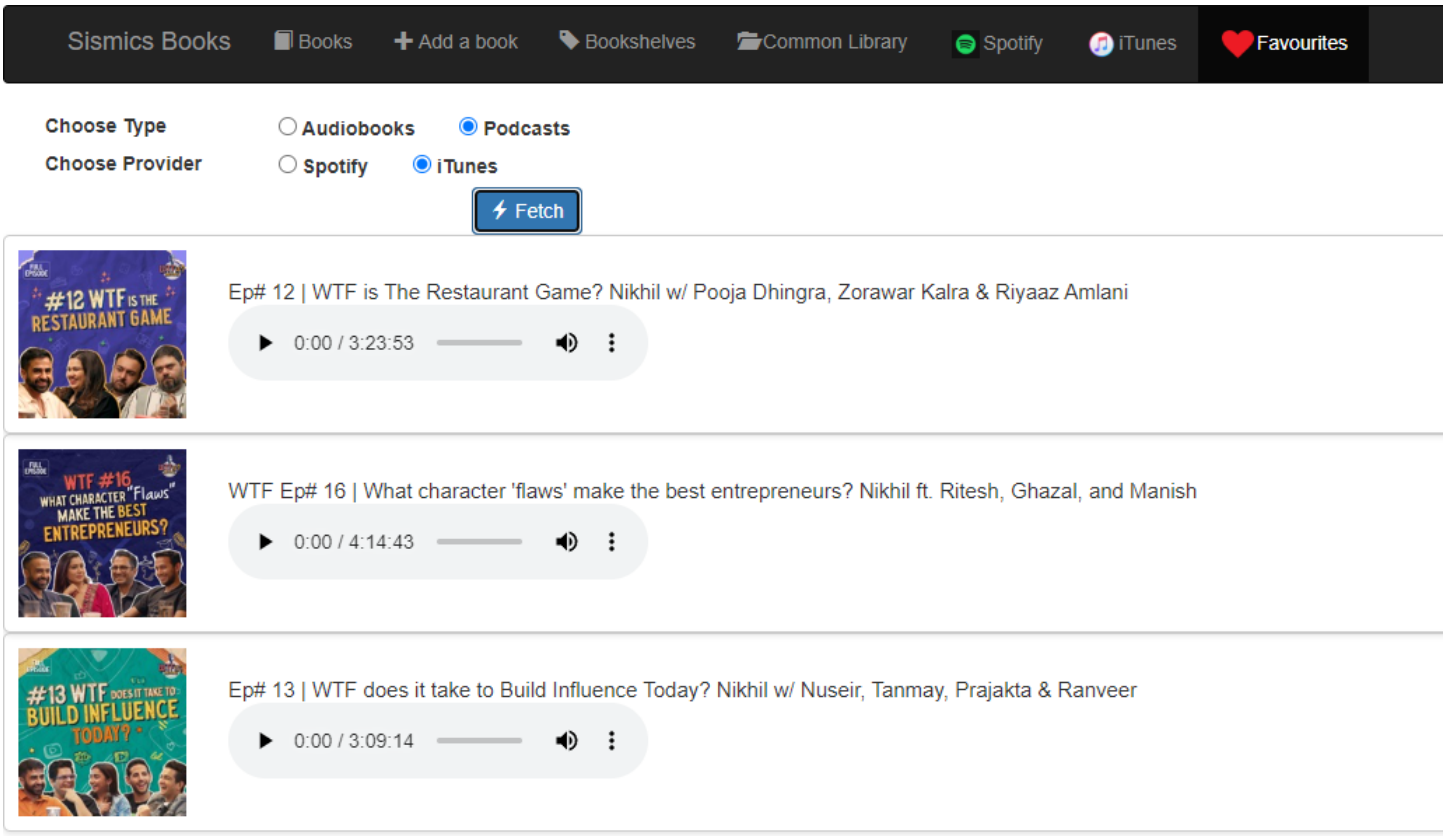
Fetching audiobooks from iTunes:



Adding audiobooks and podcasts in favourites:



Result after adding music in favourites section:



Contribution of team members :

1. Gunank Singh Jakhar -

- Online integration of Spotify and iTunes

2. M Elamparithy -

- Enhance user management system (Registration and Login)
- User self-registration

3. Rishabh Gupta -

- Book Ranking
- Book Filtering

4. Piyush Singh -

- Bonus Task (Public and Private Book self)

5. Prashant Kumar -

- Common Library Implementation(Adding a book, Common library display subsystem, Rating)