CS6.401 Software Engineering

Project 2

Sismics Books

TEAM_11

Sismics Books

Vilal, Hanuma, Shriom, Madan

1. Introduction

In the second phase of our course project, we aim to elevate the functionality and user experience of the Books repository by implementing key features and refining existing ones. Through the adept utilization of design patterns and Object-Oriented Programming (OOP) principles, we will enhance user management, introduce a common library system, and seamlessly integrate Spotify and iTunes services for audiobooks and podcasts. Additionally, we'll empower users with the ability to customize the privacy settings of their bookshelves, fostering a more personalized and secure reading experience.

2. System Overview

The system is a web-based application designed to facilitate the management of books, tags, and audio content. It aims to provide a user-friendly interface for users to add, edit, view, and delete books, manage bookshelves or tags, and link their audio content to their bookshelves. The application is built using AngularJS for the frontend, Java for the backend, and a relational database for data storage. It also integrates with external services like Spotify or iTunes for audio content.

2.1. Functional Requirements

- **User Management:** Users should be able to register, log in, and manage their profiles.
- **Book Management:** Users should be able to add, edit, view, and delete books, including managing book metadata and tags.
- **Tag Management:** Users should be able to create, edit, and delete bookshelves or tags to categorize their books.
- **Audio Management:** Users should be able to link their audio content (e.g., from Spotify or iTunes) to their bookshelves.
- **Settings Management:** Users should be able to manage their account settings, session settings, and preferences.

2.2. Non-Functional or Extra Functional Requirements

- **Performance:** The application should load quickly and handle a reasonable number of simultaneous users without significant degradation in performance.
- **Security:** User data should be securely stored, and user authentication should be robust to prevent unauthorized access.
- **Usability:** The application should be easy to use, with a clear and intuitive user interface.
- **Scalability:** The application should be able to scale to accommodate an increasing number of users and data.
- **Reliability:** The application should be reliable, with minimal downtime

2.3. Components and Connectors

- **Frontend:** AngularJS-based, handling user interactions and displaying data.
- **Backend:** Java-based, processing requests from the frontend, interacting with the database, and handling business logic.
- **Database:** Relational database, storing user data, book information, and settings.
- **External Services:** Integration with Spotify or iTunes for audio content.

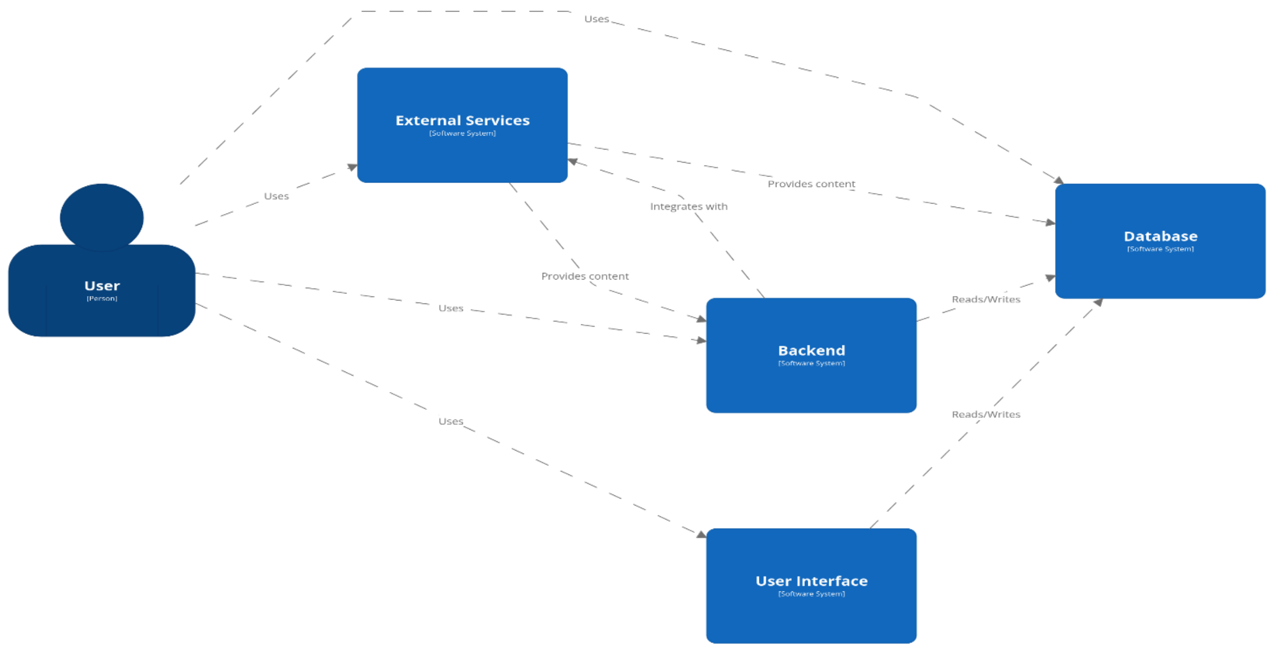
2.4. Subsystems

These subsystems are derived from the technologies and functionalities mentioned, and they represent distinct parts of the application that perform specific tasks.

Here's a breakdown:

- **User Management Subsystem**
- **Book Management Subsystem**
- **Bookshelves Management Subsystem**
- **Audio Management Subsystem**
- **Settings Management Subsystem**
- **Security and Authentication Subsystem**
- **Utility Subsystem**

2.5. System Context Diagram



[System Context] Database
Monday, 25 March, 2024 at 6:29 am India Standard Time

2.6. System Architecture

The system follows a Model-View-Controller (MVC) architecture, with a clear separation of concerns. The frontend, built with AngularJS, handles user interactions and displays data. The backend, developed in Java, processes requests from the frontend, interacts with the database, and handles business logic. The database, a relational database, stores user data, book information, and settings. The system also integrates with external services for audio content.

3. Feature 1: Better User Management:

Objective: The primary objective of this project is to enhance user experience and convenience by enabling self-user registration directly from the login page. This feature aims to streamline the registration process, making it more accessible and user-friendly.

3.1. Detailed Description:

User Registration Page

A new registration page will be introduced, allowing users to enter their username, email, and password. This page will be accessible directly from the login page, providing a seamless transition for new users.

Email Uniqueness Check

To ensure that each user has a unique email address, the system will implement a check to verify that the entered email has not been registered before. This will prevent duplicate registrations and ensure data integrity.

Password Confirmation

To enhance security and user trust, users will be required to enter their password twice. The system will compare the two entries to ensure they match. This additional step helps to prevent typos and ensures that users are aware of the password they are setting.

Validation Checks

The registration form will include several validations checks to ensure that the entered data meets the required standards:

- **Valid Email ID:** The system will validate the format of the entered email address to ensure it is correctly formatted and does not contain any invalid characters.
- **Password Requirements:** At least 8 characters long.
- These checks will help to ensure that the data entered by users is valid and secure, reducing the likelihood of errors and improving the overall security of the system.

3.2. Implementation Strategy

Frontend Development

The registration page will be developed using modern web technologies, ensuring a responsive and user-friendly interface. The form will include fields for the username, email, and password, and a confirmation field for the password. Validation checks will be implemented using JavaScript to provide immediate feedback to the user.

- **Added File:** books-web/src/main/webapp/partial/signup.html
- **Modified File:** books-web/src/main/webapp/partial/login.html
- **Modified File:** books-web/src/main/webapp/index.html
- **Modified FIle:** books-web/src/main/webapp/style/bootstrap.css

Backend Development

The backend will be responsible for handling the registration requests, including checking for email uniqueness and validating the entered data against the defined criteria. If the data is valid and the email is unique, the backend will create a new user account and store the user's information securely in the database.

- **Added File:** books-web/src/main/webapp/app/controller/Signup.js
- **Added File:** books-core/src/main/java/com/sismics/books/core/model/jpa/UserBuilder.java
- **Modified File:** books-web/src/main/java/com/sismics/books/rest/resource/UserResource.java
- **Modified File:** books-web/src/main/webapp/app/app.js

3.3. Implemented Design Pattern:

Builder Pattern:

In the above feature, we often encounter the need to create complex User objects with numerous optional parameters. To simplify this process and enhance code readability and maintainability, **we have decided to implement the Builder Pattern** for constructing User objects. The Builder Pattern separates the construction of an object from its representation, allowing flexible creation processes for different representations of the same object.

- To implement the Builder Pattern for the User class, you would create a new file named UserBuilder.java in the same package as your User class, which is likely com.sismics.books.core.model.jpa. This keeps the code organized and consistent with the existing structure.

UserBuilder.java:

```
// UserBuilder.java
package com.sismics.books.core.model.jpa;
public class UserBuilder {
        private String username;
        private String password;
        private String email;
        // ... other fields ...
        public UserBuilder setUsername(String username) {
            this.username = username;
            return this;
        }
        public UserBuilder setPassword(String password) {
            this.password = password;
            return this;
        }
        public UserBuilder setEmail(String email) {
            this.email = email;
            return this;
        }
        // ... other setter methods ...
        public User build() {
            User user = new User();
            user.setUsername(username);
            user.setPassword(password);
            user.setEmail(email);
            // ... set other fields ...
            return user;
        }
}
```

- **Using the UserBuilder:** In our UserResource class, where we often create User objects, we import UserBuilder and utilize it to construct User instances.

```
// UserResource.java
package com.sismics.books.rest.resource;
import com.sismics.books.core.model.jpa.UserBuilder;
public class UserResource extends BaseResource {
// ... existing code ...
        @PUT
        @Path("/signup")
        @Produces(MediaType.APPLICATION_JSON)
        public Response signup(
        @FormParam("username") String username,
        @FormParam("password") String password,
        @FormParam("locale") String localeId,
        @FormParam("email") String email) throws JSONException {
        User user = new UserBuilder()
            .setUsername(username)
            .setPassword(password)
            .setEmail(email)    // ... set other fields ...
            .build(); // ... existing code ...
    }// ... existing code ...
}
```

4. Feature 2: Common Library:

Objective: This feature is to create the accessibility for every used to view and contribute to library. In this the user will contribute to the library and rate the existing books. Every book which is stored in the library will have information like Title, Authors Genre(s) etc.

4.1. Detailed Description:

Book Details: First we need to make sure that every book has the below data,

- Title - Information on book title.
- Author(s) - These are to store multiple author information.
- Genre(s) (supporting multiple genres per book) - We are storing around 10 genre information in our system. Through DB insert statement.
- Rating (numerical value, calculated and averaged from user ratings) - Ratings are stored from 1 to 10.
- Thumbnail image URL (Optional) - This is optional field.

User Management: This is for multiple users to register in the system and check the books in the library. Earlier only the user who added the book could verify the books in the library.

- Adding books to the library with the required information – The points mentioned in the above book details section are taken care of.
- Rating existing books on a defined scale (1-10) - This is done based on drop down field.

This can be done by any user for a particular book.

- Viewing all books in the library – Since there are multiple users who are checking the library, it will be good to access all the books in the library.

Book Ranking: A dynamic list displays the top 10 books based on the following two criterias (will be selected by user)

- Average Rating – Average ratings of the book.
- Number of Ratings: Number of ratings given for the book.

Filtering: Users can filter displayed books by: ○

Author(s) - In our system we have stored author in comma separated format.

Genre(s) (multi-select supported) - Get the book information based on multiple genres.

Ratings (Single select, options like >6, >7, >8, >9 etc.) - Book information filtered based on average ratings.

4.2. Implementation Strategy

Frontend Development

The Common library framework is done keeping in mind the how the data needs to be viewed by the users and it should be user friendly. New fields like genre and Ratings were added in the book view screen. Below are the files which were modified related to front end.

Modified File:books-web/src/main/webapp/partial/book.edit.html

Modified File:books-web/src/main/webapp/partial/book.html

Modified File:books-web/src/main/webapp/partial/book.view.html

Backend Development

We have added new Genre and BookRating class. Authors field was increased to 1000 char to accommodate multiple authors. New class Genre was added, and this was included in book class. The combination of books and associated genre are stored in separate table. New class BookRating was added to store the rating information of user and book rated by user.

Below are the list of files where changes were done.

Added Files list :

books-core/src/main/java/com/sismics/books/core/model/jpa/BookRating.java

books-core/src/main/java/com/sismics/books/core/dao/jpa/BookRatingDao.java

books-core/src/main/java/com/sismics/books/core/model/jpa/Genre.java

books-core/src/main/java/com/sismics/books/core/dao/jpa/GenreDao.java

Modified Files list :

books-core/src/main/java/com/sismics/books/core/dao/jpa/BookDao.java

books-core/src/main/java/com/sismics/books/core/dao/jpa/UserBookDao.java

books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/UserBookDto.java

books-core/src/main/java/com/sismics/books/core/listener/async/BookImportAsyncListener.java

books-core/src/main/java/com/sismics/books/core/model/jpa/Book.java

books-core/src/main/java/com/sismics/books/core/model/jpa/User.java

books-core/src/main/java/com/sismics/books/core/service/BookDataService.java

books-core/src/main/resources/META-INF/persistence.xml

books-core/src/main/resources/db/update/dbupdate-000-0.sql

books-core/src/test/java/com/sismics/books/core/dao/jpa/TestJpa.java

books-web/src/main/java/com/sismics/books/rest/resource/BookResource.java

books-web/src/main/java/com/sismics/books/rest/resource/GenreResource.java

books-web/src/main/webapp/app/controller/Book.js

books-web/src/main/webapp/app/controller/BookAddManual.js

books-web/src/main/webapp/app/controller/BookEdit.js

books-web/src/main/webapp/app/controller/BookView.js

4.3. Implemented Design Pattern:

Strategy Pattern:

For getting the data based on average rating and number of ratings, we have to use strategy pattern.

BookRankingStrategy.java is the strategy interface to which all other concrete strategies must implement to rank books. **AverageRatingRankingStrategy.java** and **NumberOfRatingsRankingStrategy.java** are the concrete strategies for each ranking criterion (e.g., average rating, number of ratings), which implement a class that encapsulates the specific algorithm. **BookRankingResource.java** is the Controller to handle ranking requests and utilize the appropriate strategy.

Below is the **BookRankingStrategy.java** class

```
package com.sismics.books.core.strategy;

import com.sismics.books.core.dao.jpa.dto.BookRankingDto;

import java.util.List;

public interface BookRankingStrategy {
    List<BookRankingDto> rankBooks();
}
```

Below are **AverageRatingRankingStrategy.java** and **NumberOfRatingsRankingStrategy.java** class

```
public class AverageRatingRankingStrategy implements BookRankingStrategy {
    private final BookRatingDao bookRatingDao;

    public AverageRatingRankingStrategy(BookRatingDao bookRatingDao) {
        this.bookRatingDao = bookRatingDao;
    }

    @Override
    public List<BookRankingDto> rankBooks() {
        return bookRatingDao.rankBooksByAverageRating();
    }
}

public class NumberOfRatingsRankingStrategy implements BookRankingStrategy
{
    private final BookRatingDao bookRatingDao;

    public NumberOfRatingsRankingStrategy(BookRatingDao bookRatingDao) {
        this.bookRatingDao = bookRatingDao;
    }

    @Override
    public List<BookRankingDto> rankBooks() {
            return bookRatingDao.rankBooksByNumberOfRatings();
        }
    }
```

Along with these classes **BookRankingDto.java** was also added for storing the bookranking information. Frontend changes were done in book.rank.html and BookRank.js

Below are the file changes done,

Added Files list :

books-core/src/main/java/com/sismics/books/core/dao/jpa/dto/BookRankingDto.java
books-core/src/main/java/com/sismics/books/core/strategy/BookRankingStrategy.java
books-core/src/main/java/com/sismics/books/core/strategy/impl/AverageRatingRankingStrategy.java
books-core/src/main/java/com/sismics/books/core/strategy/impl/NumberOfRatingsRankingStrategy.java
books-web/src/main/java/com/sismics/books/rest/resource/BookRankingResource.java

books-web/src/main/webapp/partial/book.rank.html
books-web/src/main/webapp/app/controller/BookRank.js

5. Feature 3: Online Integration

Objective: The primary objective of this task is to enhance the user experience by integrating the selection of Spotify or iTunes as service providers for accessing audiobooks and podcasts. This integration will empower users to choose their preferred service provider and content type seamlessly within the platform, thereby enriching their overall experience and expanding the platform's functionality.

5.1. Detailed Description:

- **Selection of service providers:** Implement functionality to allow users to select either Spotify or iTunes as the service provider for accessing audiobooks and podcasts.
- **Content type selection:** Enable users to choose between audiobooks and podcasts within the selected service provider.
- **Searching and results:** Allow searching using a simple string which the user types in a search bar and then the results get displayed to the user.
- **Saving favorites:** Allow users to mark any audiobook or podcast as favorite which is then saved in the database for that user and can be accessed by the user in a separate favorites section.

5.2. Implemented Design Pattern:

Strategy Pattern Implementation

The Strategy Pattern is implemented through the AudioServiceProvider interface, which defines two methods: playAudiobook and playPodcast. This interface acts as a common interface for all audio service providers, allowing them to be used interchangeably.

- **AudioServiceProvider.java:** This interface defines the contract for all audio service providers. It has two methods: playAudiobook and playPodcast, which are expected to be implemented by any class that implements this interface.
- **SpotifyService.java and ITunesService.java:** These classes implement the AudioServiceProvider interface, providing the specific implementation for Spotify and iTunes services. Each class has its own logic for playing audiobooks and podcasts, which is encapsulated within the class.
- **AudioService.java:** This class acts as a context for the Strategy Pattern. It uses the AudioServiceProvider interface to delegate the responsibility of playing audiobooks and podcasts to the appropriate service (Spotify or iTunes). This allows the AudioService class to be decoupled from the specific implementation details of each service.

Changes in the Frontend

- **index.html:** The main HTML file that includes all the necessary JavaScript files for the application.
- **app.js:** The main AngularJS application file that sets up the application and its dependencies.
- **AudioService.js:** This AngularJS controller handles the user interactions for selecting the audio service (Spotify or iTunes), the type of content (audiobook or podcast), and the submission of the form. It uses the Restangular service to make HTTP requests to the backend, which in turn uses the Strategy Pattern to handle the specifics of each audio service.

Changes in the Backend

- AudioService.java: This REST resource class handles the HTTP requests for playing audiobooks and podcasts. It uses the AudioServiceProvider interface to delegate the responsibility to the appropriate service class (SpotifyService or ITunesService).
- SpotifyService.java and ITunesService.java: These classes implement the AudioServiceProvider interface and contain the specific logic for interacting with the Spotify and iTunes APIs

6. Project Contribution

S. N	Task Name	Implemented Design Pattern	Asignee Name	PR	Reviwer	Contribution
1	Feature 1: Better user management	Builder Pattern	Vilal	Feature-1_PR-43	Shriom	Vilal 70% Shriom 10% Team 20%
2	Feature 2: Common Library_Part-1	Strategy Pattern	Madan	Feature-2_PR-47	Vilal	Madan 70% Vilal 10% Team 20%
3	Feature 2: Common Library_Part-2	###		Feature-2_PR-48	Vilal	
4	Feature 2: Common Library_Part-3	###		Feture-2_PR-49	Vilal	
5	Feature 3: Online Integration	Strategy Pattern	Shriom	Feature-3_PR-45	Vilal	Shriom 70% Vilal 10% Team 20%
6	Bonus Feature: Public & Private Bookshelves	##	Hanuma	Bonus_PR-46	Vilal	Hanuma 70% Vilal 10% Team 20%
7	Project Report Writing	##	Vilal	Doc_Report_PR-46	Vilal	Vilal 40% Shriom 20% Hanuma 20% Madan 20%