CS6.401 Software Engineering (Spring 2024)
Project: 2, Team no: 27

Team Members

Ronak Redkar	2023201046
Vivek Ram	2023201055
Naitik Kariwal	2023201044
Karun Choudhary	2023201038
Jayank Mahaur	2023201043

Feature 1: Better User Management

Task: Enhance user experience and convenience by enabling self user registration directly from the login page.

Following are the details of the changes made in the code repository for implementing self user registration feature:

register.html:

- New Addition:
    - New user registration web page.
    - The page allows the new user to enter his/her username, email password and confirm password.
    - Checks for valid email id, at least 8 digit password and password-confirm password matching.
- File location:
    - books-web/src/main/webapp/partial/register.html

Register.js:

- New Addition:
    - Controller for User Registration page.
    - Uses backend api to save user registration details.
    - Redirects to the login page on successful registration.
- File location:
    - books-web/src/main/webapp/app/controller/Register.js

index.html:

- Modifications:
    - Added register.js script source.
- File location:
    - books-web/src/main/java/com/sismics/books/rest/resource/UserService.java

app.js:

- Modifications:
    - New route addition for Register.js controller.
- File location:
    - books-web/src/main/webapp/app/app.js

login.html:

- Modifications:
    - Added user registration page link.
    - Redirects to register.html web page when clicked.
- File location:
    - books-web/src/main/webapp/partial/login.html

UserService.java:

- Modifications:
    - Added existing user email check in register function to prevent duplicate email registration.
- File location:
    - books-web/src/main/java/com/sismics/books/rest/resource/UserService.java

UserDao.java:

- Modifications:
    - Added check for unique user email in db.
- File location:
    - books-core/src/main/java/com/sismics/books/core/dao/jpa/UserDao.java

Feature 2: Common Library

Following are the details of the changes made in the code repository for implementing Common Library System:

Task 1 - Each book in the Library should have the required information

- In order to implement the above task we have done the following -
    - We added a few properties in Book.java such as genre(to denote genre of book), common(to denote whether a book is in common library or not), thumbnailUrl(to store the url of the thumbnail fetched).
    - Created a table to store book ratings and corresponding changes to db files and persistence.xml were made.
    - Library.html - Used to display the book details such as Title , Author, Genre, Rating, Thumbnail.
    - Library.js - Implemented the method using Restangular js in order to make the api call to fetch the books from the database .

Task 2 - User Management

In order to implement the above task we have added the following so that user can interact with the system -

- Adding book to library with required information -
    - LibraryResource was made having the endpoints for performing the interactions with common library.
    - Bookview.html - If the book is not already present in the library then we have implemented the functionality to add the book from the UserBooks section to the Common Library.
    - BookView.js - Added a Rest Angular Controller method with name addToCommonLibrary() which is used to make the api call to the backend in order to implement the above functionality.
    - BookDao.java - Implemented a method named addToCommon() that takes bookId as argument and fetches the book by creating a query with the above BookId and for that particular book set the attribute named common to true.
    - BookResource.java - Implemented a function named addCommon() at the backend in order to interact with BookDao by creating a UserBookDao object and adding that book to the common library.

- Rating existing books on a defined scale (1-10) -
    - Book.view.html - Added the feature to rate the book as well as to update the rating of the books.
    - BookView.js - Implemented various Restangular js methods such as addRating(), updateRating(), deleteRating() to call the api endpoints at the backend.
    - BookRatingResource.java - Implemented the backend api's to add, update and delete the ratings of the given book.
    - BookRatingDao - Implemented the methods to perform the add, update, delete logic by writing the backend queries.

- Viewing all books in the library -
    - Library.html - Used to display the various books present in the library.
    - LibraryView.html - Used to display the detailed view of a particular view and also have a feature to add the book from the

common library to a used book if the book is not already present in the UserBooks.
    - Library.js - Implemented method is named as loadBooks() to make a call to the backend api to fetch the books.
    - LibraryResource.java - Implemented the backend api to fetch the list of the books with the help of BookDao object and TagDao object.

Task 3 - Book Ranking

In order to implement the above task we have made the following changes -

- Library.html - we have implemented the 2 criteria based on which we can fetch the books which are - Average Rating, Number of Rating.
- Library.js - Created Restangular js methods with name top10Rating() and top10Rated() in order to hit the api calls at the backend.
- LibraryResource.java - Implemented the backend api to fetch the books either based on top10rating or based on top10Rated.
- BookDao.java - Implemented 2 methods named getTop10Rating () and getTop10Rated() which contains the query logic to return the list of books based on the required criteria.
- Also created a strategy pattern around this functionality where the filter criteria was being resolved dynamically. Files added for this functionality were BookFilterCriteria.java, FilterTop10Rated.java, FilterTop10Rating.java.

Task 4 - Filter - In this task we have provided the feature by which the user can filter the books in the library based on 3 criteria which are - Authors(multi-select), Genre(multi-select), Ratings.
In order to implement the above task we have made the following changes -

- Library.html - Used to provide the filtering features on the web page in the UI side.

- Library.js - Implemented various Restangular js methods such as getAllAuthors(), getAllGenres(), applyFilter() to pass the data selected at the frontend to the backend apis to fetch the authors, genres of all the books present in the library and to apply the filters.
- LibraryResource.java - Implemented the backend api to return the list of the books based on the applied filters.
- BookDao.java - Implemented the getFilteredBooks() method which returns the list of books by fetching them from the database using the query based on the applied filter values.

Feature 3: Online Integration

Task 1: Selection of service providers
Task 2: Content type selection
Task 3: Searching and results
Task 4: Saving favorites

Following are the details of the changes made in the code repository for implementing above tasks related to online integration for audiobooks and podcasts.

AudioBookSearchResource.java

- New Addition:
    - Include api endpoint '/audiobook-search' used for searching audiobooks and podcasts.
    - Uses Strategy and Adapter Design patterns implemented to search and display results of Spotify and iTunes Api calls.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/AudiobookSearchResource.java

SpotifyAuth.java

- New Addition:
    - Include code to make OAuth 2.0 Spotify Api call to get access token which is used for authorization in other Spotify api calls.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/SpotifyAuth.java

SpotifySearch.java

- New Addition:
    - Includes code to make Spotify Search api call.
    - Uses query: search term entered by the user and itemType (audiobook/podcast) to search content.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/SpotifySearch.java

Audiobook.js

- New Addition:
    - Controller to handle api response for both Spotify and iTunes
    - Passes user entered parameters to backend code to make api calls and returns the response received to the webpage for displaying content.
- FIle Location:
    - books-web/src/main/webapp/app/controller/Audiobook.js

app.js

- Modifications:
    - Contains routes for controllers used in searching audiobooks and podcasts (audiobook.js) and to show user favorites (favourites.js)
- File Location:
    - books-web/src/main/webapp/app/app.js

audiobook.html

- New Addition:
    - Web page where user interacts to search for audiobooks and podcasts.
    - Displays search results to users.
- File Location:
    - books-web/src/main/webapp/partial/audiobook.html

index.html

- Modifications:
    - Added reference to new controllers.
- File Location:
    - books-web/src/main/webapp/index.html

iTunesSearch.java

- New Addition:
    - Logic to search for audiobooks or podcasts based on user inputs using iTunes API call.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/iTunesSearch.java

SearchStrategy.java

- New Addition:
    - Implementation of Search Strategy pattern for iTunes and Spotify.
- FIle Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/SearchStrategy.java

UserFavouriteResource.java

- New Addition:
    - Includes api endpoints ‘/userfavourite’ and ‘/show’ used for fetching user added favorite audiobooks/podcasts from the database and to show them to the user.
- FIle Location:

    - books-web/src/main/java/com/sismics/books/rest/resource/UserFavouriteResource

UserFavouritesDao.java

- New Addition:
    - Data Access Object logic for updating and reading User Favorite table from database.
- File Location:
    - books-core/src/main/java/com/sismics/books/core/dao/jpa/UserFavouritesDao.java

UserFavourites.java

- New Addition:
    - Model for User Favorite table.
- File Location:
    - books-core/src/main/java/com/sismics/books/core/model/jpa/UserFavourites.java

Favourites.js

- New Addition:
    - Controller that makes api calls to fetch and show users added favorite audiobooks and podcasts.
- File Location:
    - books-web/src/main/webapp/app/controller/Favourites.js

Favourites.html

- New Addition:
    - Webpage to display user favorite audiobooks and podcasts.
- File Location:
    - books-web/src/main/webapp/partial/favourites.html

ResponseAdapter:

- New Addition:
    - Includes Response Adapter Interface.
- File Location:

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

iTunesResponseAdapter.java
- New Addition:
    - Implements Adapter Interface for iTunes.
    - Used to adapt to response received for both audiobooks and podcasts.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/iTunesResponseAdapter.java

SpotifyResponseAdapter.java
- New Addition:
    - Implements Adapter Interface for Spotify.
    - Used to adapt to response received for both audiobooks and podcasts.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/SpotifyResponseAdapter.java

iTunesFetch.java
- New Addition:
    - Logic to search for multiple audiobooks and/or podcasts which are user favorites using iTunes api call.
- File Location:
    - books-web/src/main/java/com/sismics/books/rest/resource/iTunesFetch.java

SpotifyFetch.java
- New Addition:
    - Logic to search for multiple audiobooks and/or podcasts which are user favorites using Spotify Get Audiobooks and Get Episodes api calls.
- File Location:

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

dbupdate.sql

- Modification:
    - Added a new table to save User favorites data in the database.
- File Location:
    - books-core/src/main/resources/db/update/dbupdate-000-0.sql

Design Patterns Implemented:

- Strategy Design Pattern:
    - Used to define a family of algorithms, encapsulate each one, and make them interchangeable. In this case, it is used to select different search strategies based on the service provider such as Spotify or iTunes.
    - Interface SearchStrategy:
        - Defines a common method to perform the search operation.
        - Concrete implementations - SpotifySearchStrategy and iTunesSearchStrategy: Implements the SearchStrategy interface and provides specific search logic for Spotify and iTunes.

- Adapter Design Pattern:
    - Used to adapt the interface of a class into another interface that a client expects. In this case, it adapts the responses received from Spotify and iTunes APIs to a common format that the frontend can handle uniformly.
    - The responses from service providers such as Spotify and iTunes are different in structure for searching. Within each Service Provider, the responses differ for different content types such as audiobooks and podcasts. This makes the use of adapter pattern appropriate.
    - Interface ResponseAdapter: Defines a method ‘adapt’ to transform the response format.

    - Concrete implementations SpotifyResponseAdapter and iTunesResponseAdapter: Implement the ResponseAdapter interface and provide logic to adapt the responses from Spotify and iTunes APIs.

Project Contributions:
Vivek Ram:
Feature 2-

- Book Details:
    - Made backend changes to use Book.java to also store genre, thumbnailUrl.
    - Made BookRating table to store the ratings given by users to each books.
    - Made api endpoints - bookrating/ to cater the needs of adding, updating, deleting, etc the book ratings.
    - Added UI capabilities to use these backend features and display the content on UI.
- User Management:
    - Made backend changes to allow users to interact with common library, enabling him to add books to common library, give ratings to books, and viewing all the books from common library.
    - Made api endpoints - book/id:/addcommon, library/id:/remove, library/id:/adduserbook, library/id:/hasuserbook to enable the ability to add/remove the books from common library, also an api to add a book directly from common library to user library and to check whether the user has the book in his user books.
    - Added UI capabilities to use these backend features and display the content on UI.

Jayank Mahaur:
Feature 2 - Task3 - Book Ranking

- Updated the frontend UI to provide the functionality to display the top 10 books based on average rating, number of rating.

- Added the files in the controller to add Restangular js methods to communicate to the backend apis to perform the desired functionality based on the chosen feature at the frontend.
- Added methods like getTop10Rating() and getTop10Rated() to communicate to the backend apis for the desired functionality.
- Created api endpoints - library/top10 to connect to backend to fetch the books based on the chosen criteria.
- Implemented the Strategy Pattern for filtering based on the filtering criteria given i.e. top 10 books based on average rating, top 10 books based on number of ratings.

Ronak Redkar:

- Feature 1: Self User Registration
    - Updated code in backend (UserResource.java, UserService.java and UserDao.java) to enable and validate new user registration.
- Feature 3: Online Integration
    - Developed complete backend logic and code for API endpoints such as Spotify search api for audiobooks and podcasts, handling API response, database activities for user added favorite items. (AudioBookSearchResource.java, SpotifyAuth.java, SpotifySearch.java, UserFavouriteResouce.java, UserFavouritesDao.java, UserFavourites.java, iTunesResponseAdapter.java, SpotifyResponseAdapter.java)
    - Implemented Adapter Design Pattern to handle search results of API calls and communicate with the frontend.
    - Implemented backend code to add User favorite audiobooks/podcasts in database and fetch from database to display to user via api endpoints.
    - Added logic and code to make service provider api calls (Spotify - Get Several Audiobooks, Spotify - Get Several Episodes and iTunes Search) for fetching user favorite items using item Ids fetched from the database for the specific user.
    - Tested working of Api calls and code integration.

Naitik Kariwal:

Feature 2 - Task4 - Filtering

- Updated the frontend UI to implement the filtering feature based on the 3 features - authors, genres and rating.
- Written Restangular js methods to make the api call to pass the frontend selected data of the authors, genres and rating for all the books present in the library.
- Build the back end api structure in order to fetch all authors using library/authors, all genres using library/genres.
- Also made library/filters fetch the books based on the filtering conditions provided and library/list to list down all the books which are present in the common library.

Karun Choudhary

Feature 1: Self User Registration

- Create frontend page for registration
- Handle controller to pass data accordingly to required class.

Feature 2:Online integration

- Developed code for iTunes search api call to show audiobooks/podcasts to the users.
- Implemented strategy pattern for the itunes and spotify search accordingly to their need.
- Created web pages (audiobook.html and favorites.html) to display search results and user favorite items to users.
- Added controllers (audiobook.js and favorites.js) to handle backend api response and communication.
- Handled and created files for implementing strategy pattern AudioBookSearchResource.java , itunessearch.java, SpotifySearch.java, Search.Strategy.java.