

# SE Project-2 Report

## Table of Contents

- [Table of Contents](#)
- [Team Members](#)
- [Feature 1 : Better User Management](#)
  - [Introduction](#)
  - [Plan](#)
  - [Design Patterns](#)
    - [Chain of Responsibility](#)
  - [Implementation Detail](#)
    - [Front End](#)
    - [Back End](#)
  - [User Interface Pictures](#)
- [Feature 2 : Common Library](#)
  - [Introduction](#)
  - [Plan](#)
  - [Design Patterns](#)
  - [Implementation Details](#)
    - [Front End](#)
    - [Back End](#)
  - [User Interface Pictures](#)
- [Feature 3 : Online Integration](#)
  - [Introduction](#)
  - [Plan](#)
  - [Design Patterns](#)
    - [Factory Pattern](#)
    - [Adapter Pattern](#)
  - [Implementation Details](#)
    - [Front End](#)
    - [Back End](#)
  - [User Interface Pictures](#)

## Team Members

- 1) Vineeth Bhat (2021101103)
- 2) Ishwar B Balappanawar (2021101023)
- 3) Swayam Agrawal (2021101068)
- 4) Mitansh Kayathwal (2021101026)
- 5) G.L.Vaishnavi (2023204009)

## Feature 1 : Better User Management

### Introduction

Our app has a user management system already in place, which you have already looked at in detail. The current flow of creating a user involves logging into an admin account and creating a new user manually. However, this is neither intuitive nor convenient, and thus demands an improvement.

**Objective:** We wish to enhance user experience and convenience by enabling self user registration directly from the login page.

### Plan

We first addressed the issue with the broken Nav Bar, which previously displayed an option to Logout even before a user had logged in.

To improve user experience during registration, we decided to introduce a "Sign Up" tab in the Nav Bar. This addition aims to make the registration process more intuitive and accessible for users.

When users click on the Sign Up tab, they are directed to a user interface (UI) similar to the Login page for consistency. The registration page requires users to enter their username, email address, and password with the following specifications:

- Email uniqueness check: The system verifies that the entered email has not been registered before.
- Password confirmation: Users are prompted to enter their password twice for confirmation.
- Validation checks: The system enforces checks such as valid email format, a minimum password length of 8 characters, etc.

Both the Login and Sign Up processes feature robust error handling mechanisms that provide clear warnings to users about any incorrect inputs.

The Sign Up button remains disabled until all validation requirements are met. This measure not only ensures data integrity but also prevents any invalid requests from reaching the backend server.

Upon successful registration, users are automatically redirected to the Login page to access their newly created accounts.

## Design Patterns

### Chain of Responsibility

We leveraged the Chain of Responsibility pattern to refactor the `validateSecurePassword` function, which previously relied on multiple validation functions for different password criteria. Here are the steps we took to implement this pattern:

#### 1. Handler Interface:

- We created the `PasswordHandler` interface, serving as a base abstract class, or interface, for all handlers involved in the validation process.

#### 2. Concrete Handler Classes:

- Multiple concrete handler classes were implemented, each responsible for validating a specific aspect of the password (e.g., length, presence of digits, lowercase characters, uppercase characters, special characters).
- Each handler class adheres to the `PasswordHandler` interface and throws an exception if the specific validation rule is not met.
- Implemented concrete handler classes ( `LengthHandler` , `DigitHandler` , `LowercaseHandler` , `UppercaseHandler` , `SpecialCharHandler` ) that implement the `PasswordHandler` interface.

#### 3. Chain Construction:

- The handlers were chained together to form a validation pipeline, where each handler processes the password according to its validation rule.

This refactoring effort seamlessly integrated into our existing codebase, enhancing maintainability and extensibility.

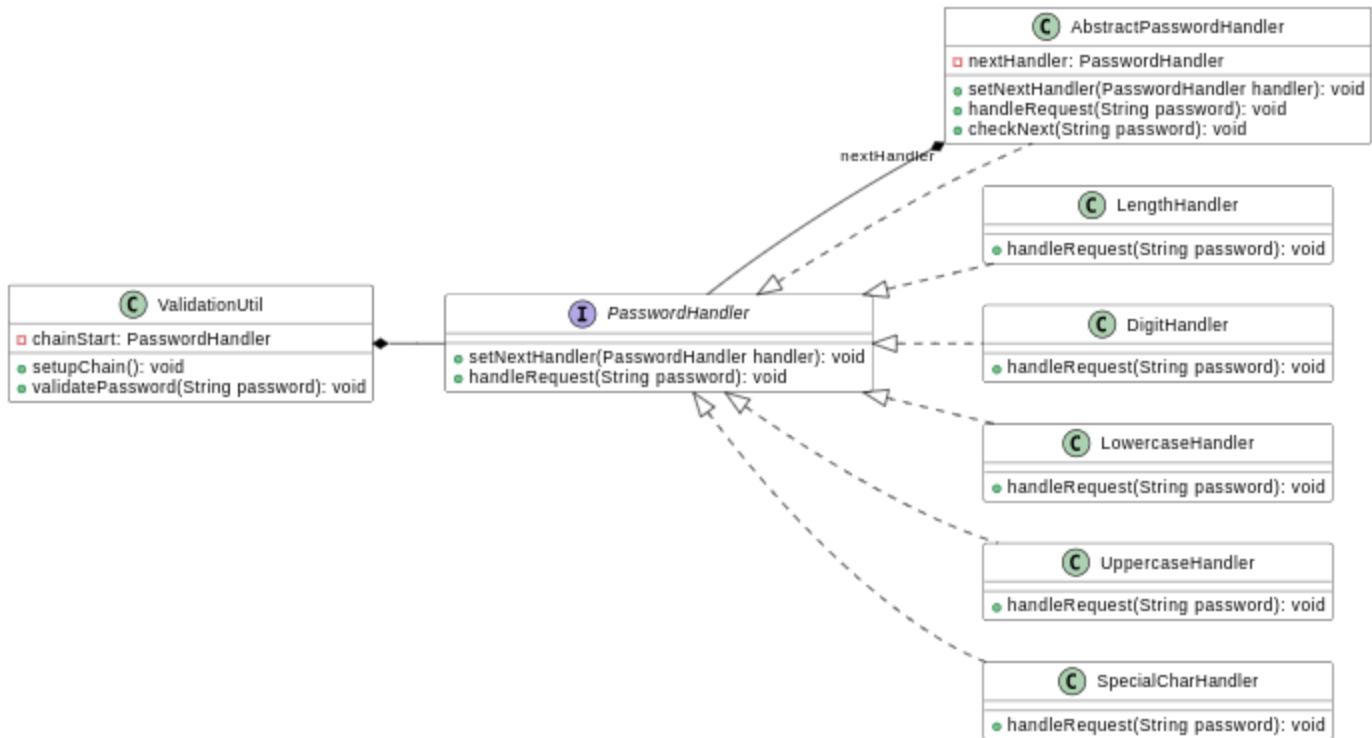
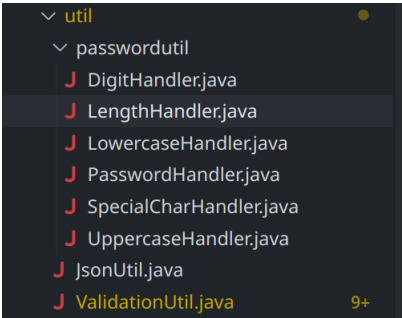
In summary, the Chain of Responsibility pattern facilitated a modular and organized approach to password validation, improving code readability and scalability in managing different validation rules.

Overall, these changes encapsulate each password validation rule in separate handler classes, adhering to the principles of the Chain of Responsibility pattern. The validation pipeline ensures that each rule is checked sequentially, and validation stops at the first rule violation, throwing an appropriate exception.

```
/**
 * Password Validate.
 *
 * @param s String to validate
 * @param name Name of the parameter
 * @throws JSONException
 */
public static void validateSecurePassword(String s, String name) throws JSONException {
    PasswordHandler lengthHandler = new LengthHandler();
    PasswordHandler digitHandler = new DigitHandler();
    PasswordHandler lowercaseHandler = new LowercaseHandler();
    PasswordHandler uppercaseHandler = new UppercaseHandler();
    PasswordHandler specialCharHandler = new SpecialCharHandler();

    lengthHandler.setNextHandler(digitHandler);
    digitHandler.setNextHandler(lowercaseHandler);
    lowercaseHandler.setNextHandler(uppercaseHandler);
    uppercaseHandler.setNextHandler(specialCharHandler);

    lengthHandler.handleRequest(s, name);
}
```



## Implementation Detail

### Front End

- signup/html
- Signup.js

These files were added to handle the front end part. Including the Sign Up page and functionalities to perform the necessary requests to the server for the registration of the signed up user.

### Back End

- UserHandler.java
- ValidationUtil.java
- passwordutil directory

The APIs necessary for this functionality are handled in UserHandler.java. ValidationUtil Class and the files in passwordUtil directory are used for validation.

## User Interface Pictures

Sismics Books

Login

Sign Up

Username

Username

Username is required.

Username must be at least 3 characters long.

E-mail

vj

Invalid email format.

Password

\*\*\*\*\*

Password must have at least 8 characters.

Password must have at least 1 special character.

Password must have at least 1 uppercase letter.

Password must have at least 1 digit.

Password (confirm)

\*

Passwords do not match.

Sign up

Sismics Books

Login

Sign Up

Username

ishwar

E-mail

ishwarbb23@gmail.com

Password

\*\*\*\*\*

Password (confirm)

\*\*\*\*\*

Sign up

## Feature 2 : Common Library

### Introduction

How about a common library system accessible to all users, allowing them to contribute books, rate existing books, and explore the library’s collection!

**Objective:** Implement a common library accessible to all users for contributing and exploring books. This feature allows users to add books in a common library and rate the books.

### Plan

Leveraging the documentatioin done in the previous assignment, the initial work comprised of designing new schemas for the necessary work. We added SQL statements and relations for libray books, ratings and genres.

Subsequently, we defined the DAO and DTO classes for all of the classes made and a criteria class for the library books (explained in the implementation details section). These classes were populated with the necessary attributes, getters, setters and other methods to retrieve relevant objects.

Finally, we defined several new API calls within the backend and conducted thorough API testing using Postman to make sure that our code logic and validation parameters were functioning aptly.

Meanwhile, the frontend was developed concurrently through making pre-defined API calls that returned dummy data.

Given the API calls, we planned to keep the UI consistent across all pages (including the ones which were defined earlier).

We initially planned to create three new pages. The first page will be defined by `library.html` and would show all the books in the common library in a bookshelf format. We also planned to add options to filter and rank books with the options specified in the project description here. On this page, there is also a button to add books to the common library, selecting on which the user will be redirected to the second page defined below.

The second page provided the user with the option to add a book to the library by giving the ISBN number of the book as an input. Since the project description did not provide explicit requirement pertaining to adding books in a specific format, we did this implementation based on our assumption, the user knows the ISBN of the book he/she wants to add. This page is defined by `library.addBook.html`

The third page will be defined by `library.bookDetail.html` which would show the details of a book when one is selected from the previously defined page. Here we planned to provide an option for the user to add genre and provide rating of the corresponding book. All validations were handled here.

### Design Patterns

#### Strategy Pattern

We’ve implemented two strategy patterns - the Sorting Strategy Pattern and the Filtering Strategy Pattern. These patterns are used to encapsulate algorithms within interchangeable components, allowing for flexibility and easy

modification of behavior without altering the core logic.

The Sorting Strategy Pattern comprises the `LibBookSortStrategy` interface, defining a common method for sorting a list of `LibBookDto` objects. Concrete sorting strategies, such as `AverageRatingSortStrategy` and `NumRatingsSortStrategy`, implement this interface, each encapsulating a specific sorting algorithm based on average rating and number of ratings, respectively. The `LibBookSortContext` acts as the context in which sorting strategies are applied, allowing for dynamic selection and execution of sorting algorithms. This pattern was implemented to decouple the sorting logic from the main `findByCriteria` code to promote reusability and maintainability.

▼ java / com / sismics

▼ books / core

> constant

> dao

> file

▼ jpa

> criteria

> dto

J AudioBookDao.java

J AuthenticationTokenDao.java

J BookDao.java

J ConfigDao.java

J LibBookDao.java

J LibBookGenreDao.java

J LibBookRatingDao.java

J LocaleDao.java

J PodcastDao.java

J RoleBaseFunctionDao.java

J TagDao.java

J UserAppDao.java

320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343

```
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Similarly, the Filtering Strategy Pattern encompasses the `LibBookFilterStrategy` interface, defining a method for determining whether a `LibBookDto` object satisfies certain filtering criteria. Concrete filtering strategies, such as `AuthorFilterStrategy`, `GenreFilterStrategy`, and `RatingFilterStrategy`, implement this interface, each encapsulating a specific filtering criterion based on author, genre, or minimum rating. The `LibBookFilterContext` acts as the context for applying filtering strategies, enabling the application of multiple filtering criteria to a single `LibBookDto`. This pattern was implemented to modularize filtering logic and enhance resuability in the event of new filtering logic.

> dev

▼ main

▼ java / com / sismics

▼ books / core

> constant

> dao

> file

▼ jpa

> criteria

> dto

J AudioBookDao.java

J AuthenticationTokenDao.java

J BookDao.java

J ConfigDao.java

J LibBookDao.java

J LibBookGenreDao.java

J LibBookRatingDao.java

J LocaleDao.java

J PodcastDao.java

J RoleBaseFunctionDao.java

J TagDao.java

J UserAppDao.java

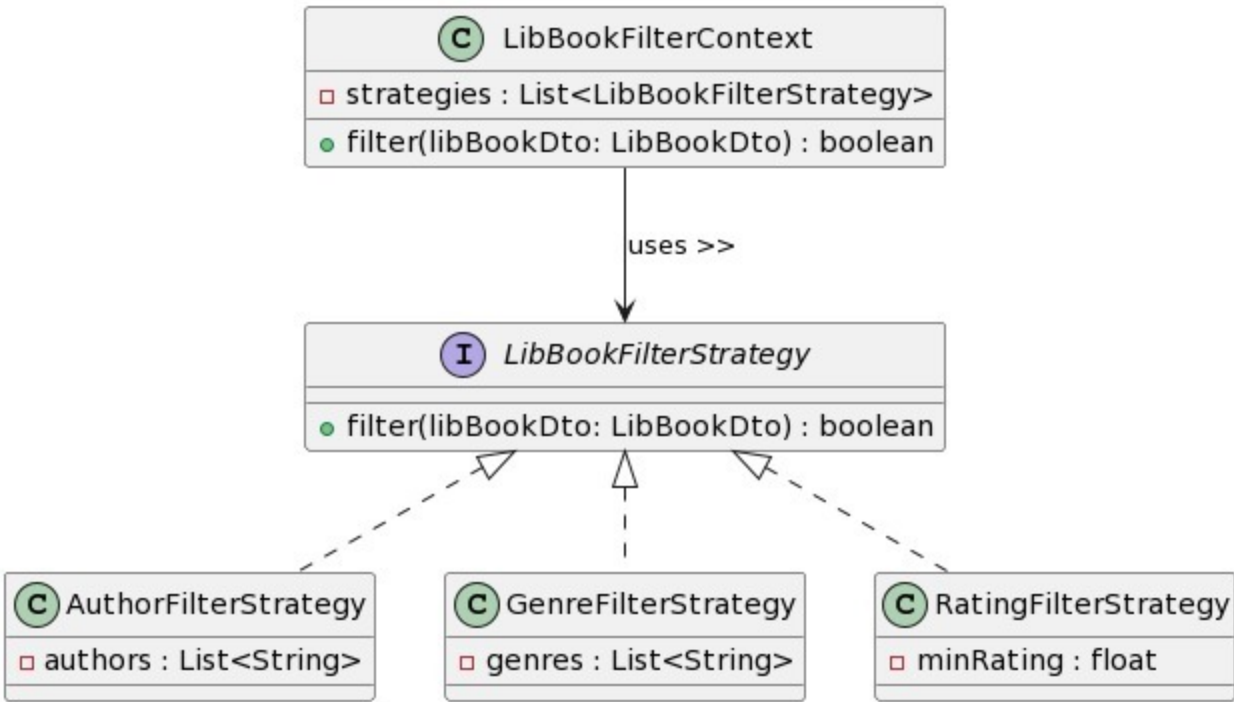
J UserAudioBookDao.java

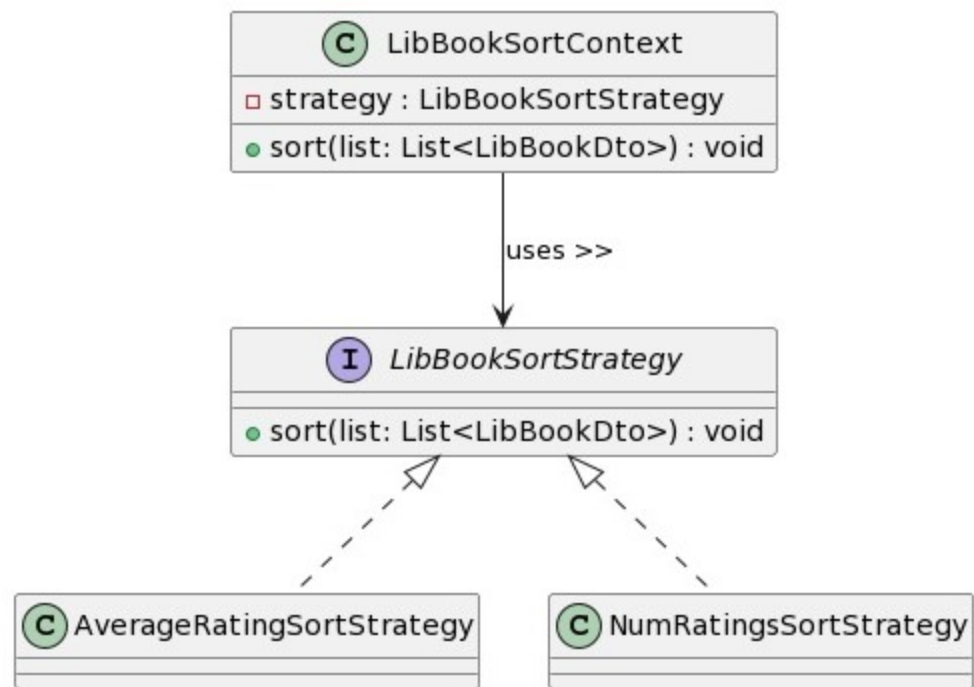
J UserBookDao.java

J UserContactDao.java

198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226

```
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```





## Implementation Details

### Front End

#### **library.html** - Structure and Functionality:

The **library.html** page serves as the main interface for users to interact with the library's collection. It's structured into several key sections:

1. **Add Book Button:** A link to navigate users to a page where they can manually add a new book to the library.
2. **Filter Books Button:** Users can filter the library's collection based on authors, genres, and ratings (multi-select supported).
3. **Rank Books Button:** Users can rank the books by average rating or by the number of ratings.
4. **Books Section:** Displays the books currently available in the library. Each book is shown with its cover image, title, and author. Clicking on a book navigates to its detailed view.

The page uses Angular directives such as **ng-click** for handling user interactions like toggling filters/ranking and navigating to book details, **ng-repeat** for dynamically listing books and genres, and **ng-if** for conditionally rendering sections based on the state (e.g., showing filters or ranking options).

**LibraryController.js** is responsible for managing the state and interactions on the **library.html** page.

- The **fetchBooks()** function defined here constructs a query with specified queryParams with filters and ranking criteria selected by the user and calls the backend API (**library/list**) using Restangular. The books fetched based on these criteria are then stored in the **\$scope.books** array for display. If ranking is applied, a limit is set to fetch the top 10 books.

**LibraryAddBook.js** is the controller for the page responsible to add books to the common library. We simply do a backend API call to add the entry to the table.

**LibraryBookDetailController.js** manages the interaction and state of a detailed book view in a library application.

This controller is responsible for fetching detailed information about a specific book, allowing users to add genres to the book, and enabling users to rate the book. We have pre-defined the genres which can be added to a book to ensure integrity of the application and this is one of our assumptions.

### Back End

We defined tables **T\_LIB\_BOOK**, **T\_LIB\_BOOK\_GENRE** and **T\_LIB\_BOOK\_RATING** within **books-core/src/main/resources/db/update/dbupdate-000-0.sql** and wrote the corresponding models within the **books-core/src/main/java/com/sismics/books/core/model/jpa** directory.

Further, we defined the DAO classes within **books-core/src/main/java/com/sismics/books/core/dao/jpa** and the corresponding DTO classes in **books-core/src/main/java/com/sismics/books/core/dao/jpa/dto** directory. Methods corresponding to deleting a book, creating a library book, getting a book by its ID were defined inside **LibBookDao.java** with corresponding codes for the ratings and genres being defined within the **LibBookRatingDao.java** and **LibBookGenreDao.java** files.



The `LibBookGenreDao.java` defined methods such as `getByBookIdAndName()` , `getByLibBookId()` and so on that are used to retrieve the `LibBookGenre` objects based on the given parameters. Similarly, we have functions that retrieve specific objects or a class of objects `LibBookRatings.java` .

The `LibBookDao.java` also defines a `findByCriteria()` that takes in a list of author names, list of genre names and a minimum rating and uses strategy pattern to filter it as mentioned earlier - `LibBookFilterContext` . The function also does sorting based on the `LibBookSortContext` class.

We define the necessary APIs:

#### `AddLibBook` API

- **Path:** `/library`
- **Method:** PUT
- **Functionality:** Adds a new book to the library. It validates the provided ISBN, fetches book details (potentially from a public API if not found locally), and saves the book in the database. It also checks if the book is already added to the library to prevent duplicates.
- **Parameters:**
  - `isbn` : The ISBN of the book to add.
- **Authentication:** Requires user authentication to execute.

#### `AddLibBookGenre` API

- **Path:** `/library/genre`
- **Method:** PUT
- **Functionality:** Associates a new genre with a book in the library. It validates the existence of the specified library book and ensures that the genre is not already associated with the book.
- **Parameters:**
  - `libBookId` : The unique identifier of the library book.
  - `genreName` : The name of the genre to add.
- **Authentication:** Requires user authentication to execute.

#### `AddLibBookRating` API

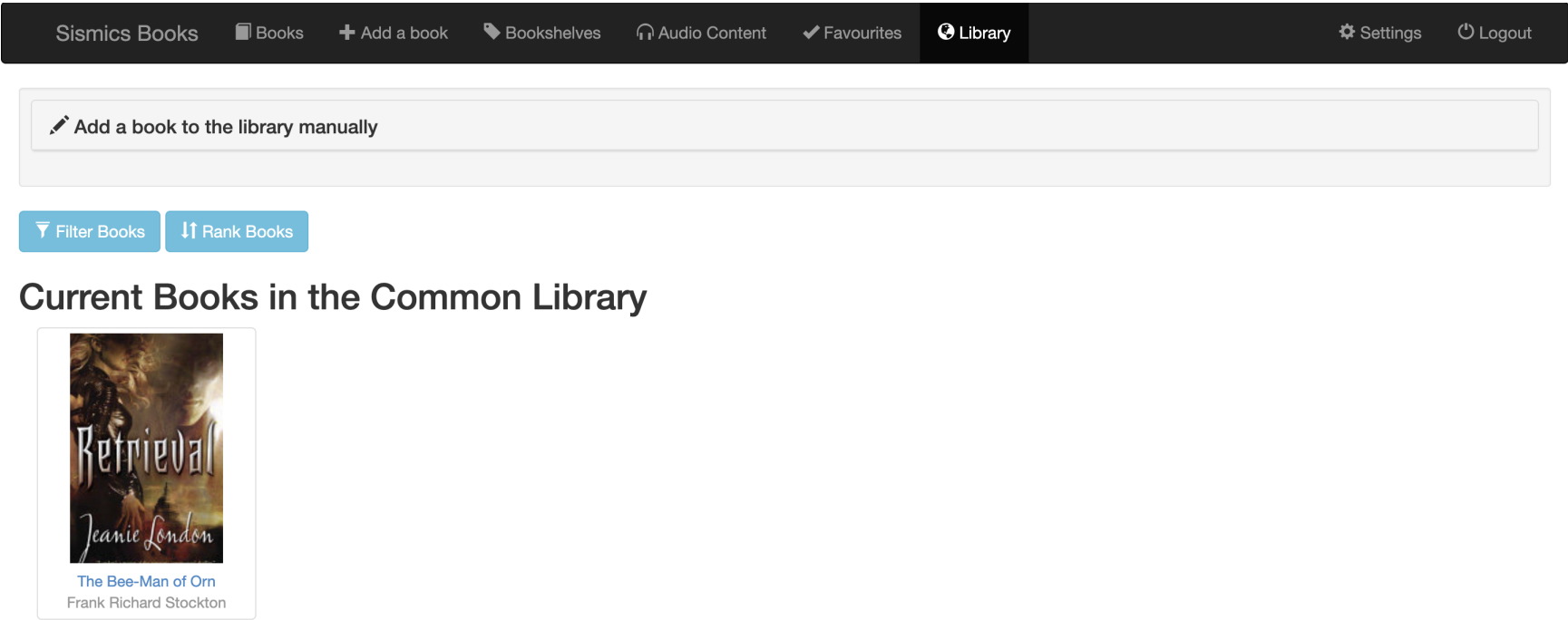
- **Path:** `/library/rating`
- **Method:** PUT
- **Functionality:** Allows users to rate a book in the library. It ensures that the book exists and that the same user has not already rated the book. The rating value is expected to be within a specified range.
- **Parameters:**
  - `libBookId` : The unique identifier of the library book.
  - `userId` : The identifier of the user submitting the rating.
  - `rating` : The rating value.
- **Authentication:** Requires user authentication to execute.

#### `ListLibBooks` API

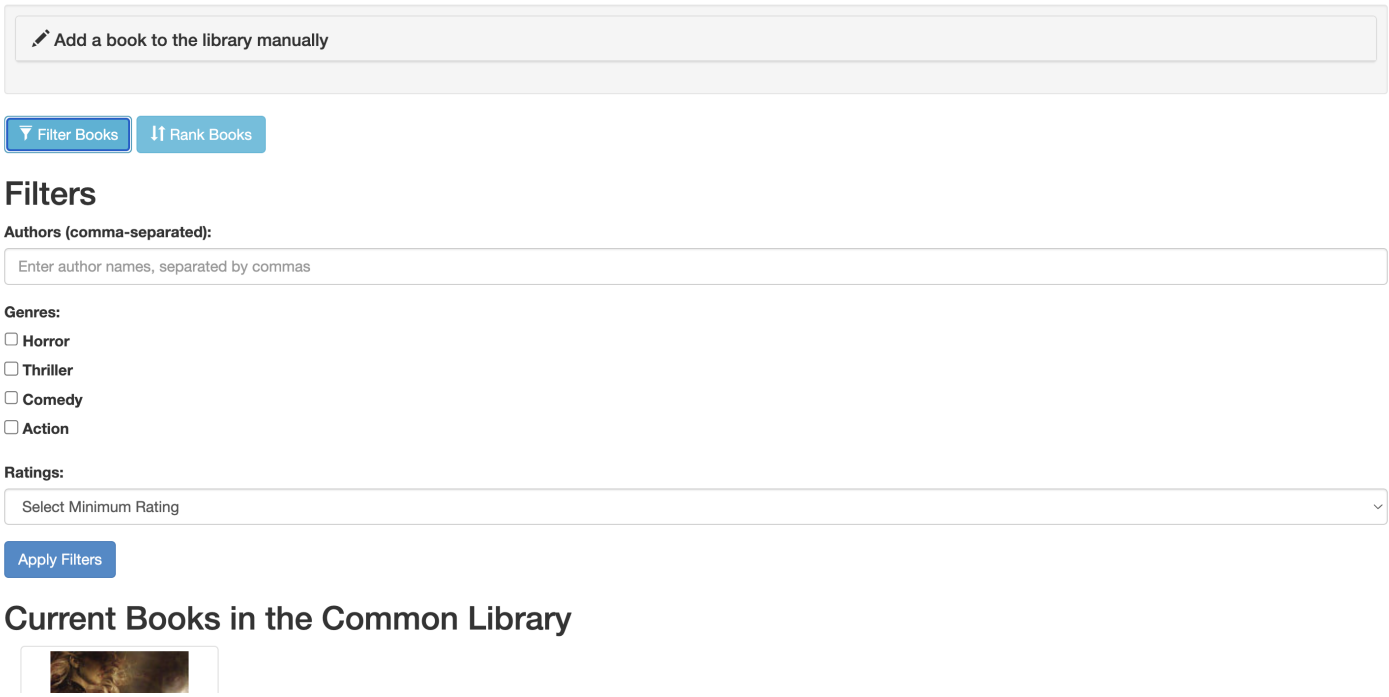
- **Path:** `/library/list`
- **Method:** GET
- **Functionality:** Retrieves a list of books from the library, optionally applying filters for authors, genres, minimum ratings, and sorting criteria. It supports pagination through `limit` and `offset` parameters and allows sorting by average rating or number of ratings.
- **Parameters:**
  - `limit` , `offset` : Pagination controls.
  - `sort_column` , `asc` : Sorting controls.
  - `search` : A search query to filter books by title, subtitle, or author.

- `filter_authors` : Comma-separated list of author names to filter books.
  - `filter_genres` : Comma-separated list of genres to filter books.
  - `filter_min_rating` : Minimum rating filter.
  - `library_sort` : Controls the sorting of books based on rating criteria.
- Authentication:** Requires user authentication to execute.

## User Interface Pictures



Option to filter the books present in the library:



Option to rank the books present in the library:



Sismics BooksBooksAdd a bookBookshelvesAudio ContentFavouritesLibrarySettingsLogout

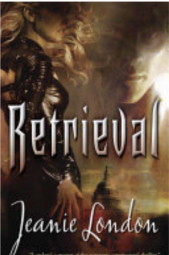
Add a book to the library manually

Filter BooksRank Books

Ranking Criteria

By Average RatingBy Number of Ratings

Current Books in the Common Library



The Bee-Man of Orn

Frank Richard Stockton

Option to add books to the library:

Sismics BooksBooksAdd a bookBookshelvesAudio ContentFavouritesLibrarySettingsLogout

Add a book to the library

ISBN:

Enter book ISBN

Submit

Cancel

Option to view book details, rate it on a scale (1-10) and add genres:

SE Project-2 Report

9

# The Bee-Man of Orn Frank Richard Stockton



Using her unique talents to intervene in the lives of humans to help them resist temptation and avoid untimely deaths, Nina, a guide between the living and the dead, finds herself caught in the middle between good and evil forces battling for control of Purgatory and battling her own temptations. Original.

Genres: Action, Comedy, Horror, Thriller  
Average Rating: 0  
Number of Ratings: 0

Add Genre

### Select Genres to Add:

- ☐ Action
- ☐ Thriller
- ☐ Comedy
- ☐ Horror

Add Selected Genres

### Rate this Book:

5 Submit Rating

## Feature 3 : Online Integration

### Introduction

Ever wished your library could groove to your favorite tunes or narrate your go-to podcast? Say hello to our latest feature: integrating Spotify and iTunes right into our platform!

**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

### Plan

#### API Testing on Postman

During our development process, we conducted thorough API testing using Postman to understand and familiarize ourselves with the response contents and formats provided by Spotify and iTunes. We observed a key difference in the responses from these platforms:

- Spotify's responses lacked a description field, unlike iTunes.

#### Adapter Pattern Implementation

Due to the significant differences in the data structures of audiobook and podcast objects between iTunes and Spotify, we opted to create a common representation for both using the Adapter pattern. This decision facilitated easier handling and processing of data from both providers.

#### UI Design Decision: Mandatory User Selection before Search

We implemented a forced user selection step before initiating a search to prevent erroneous or incomplete requests. This step ensured that the necessary factories and configurations were set up correctly before making API requests, thereby reducing potential errors.

#### Direct Display of API Results and Ignoring Pagination

In our UI design, we prioritized displaying results directly as provided by the API responses. Since our focus was not extensively on frontend development, we decided to ignore pagination for simplicity and straightforwardness in displaying data.

#### Clickable Books and Consistency in UI

Clicking on a book item makes it 'alive' in the UI, allowing users to add it to their favorites seamlessly. This approach ensures consistency across the user experience and enhances UI smoothness.

### Handling Book Addition on Click

When a user clicks on a book, we first add it to our database as a cached entry. This cached entry is not immediately added to the user's specific collections (like UserAudioBooks or UserPodcasts) but is stored in separate AudioBooks and Podcasts databases. This strategy minimizes database failures and optimizes performance.

### Protection Against False Insertions

To maintain data integrity, we implemented safeguards to prevent any malicious or erroneous insertions into our databases, ensuring data reliability and security.

### Displaying Favorites and UI Navigation

We dedicated a separate page for displaying user favorites, allowing easy navigation and toggling between audio book favorites and podcast favorites. This design simplifies browsing and enhances user experience.

### Abstracting Service Providers

Our system abstracts away the differences between service providers, treating all audio content sources uniformly. This abstraction simplifies development and maintenance by encapsulating provider-specific details.

### Direct Access to Favorite Content

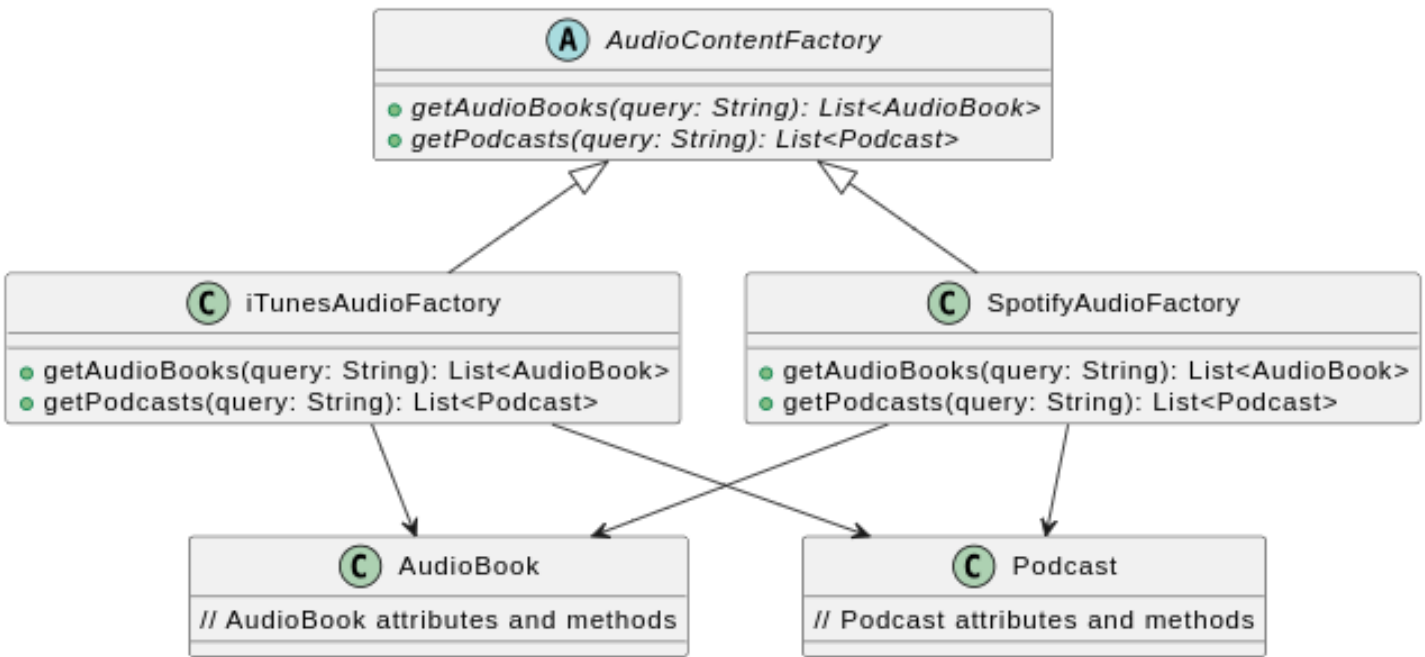
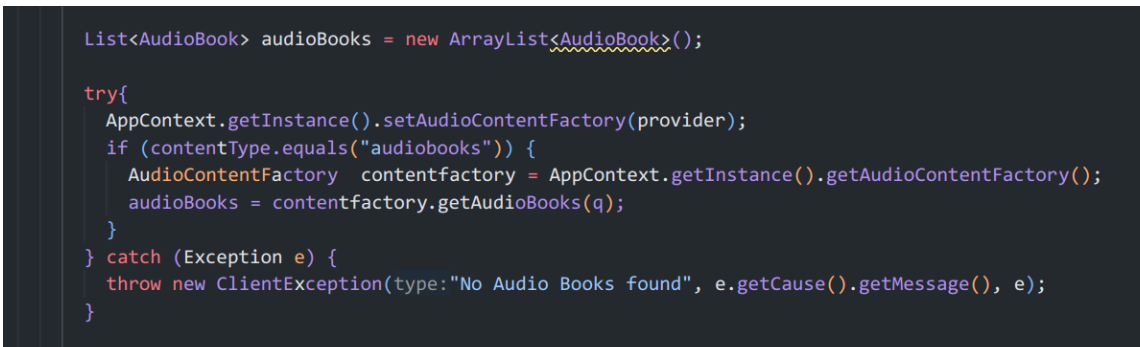
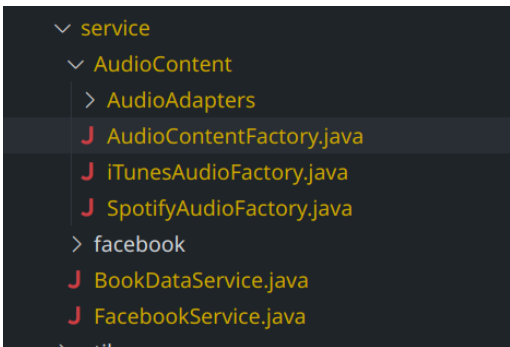
Clicking on favorite books directly accesses the corresponding audio content, leveraging the pre-stored data and eliminating the need for redundant queries, thus optimizing performance and user experience.

## Design Patterns

### Factory Pattern

We have implemented the Factory Pattern in our codebase using the AudioContentFactory. This factory includes specific implementations such as iTunesAudioContentFactory and SpotifyAudioContentFactory. Depending on the requested service provider, we set the appropriate factory in the App Context. These factories are responsible for creating products such as podcasts and audiobooks.

Our primary focus with this pattern is on extensibility, allowing for easy integration of new service providers like Youtube Music in the future. The Factory Pattern facilitates the addition of new services seamlessly.



### Adapter Pattern

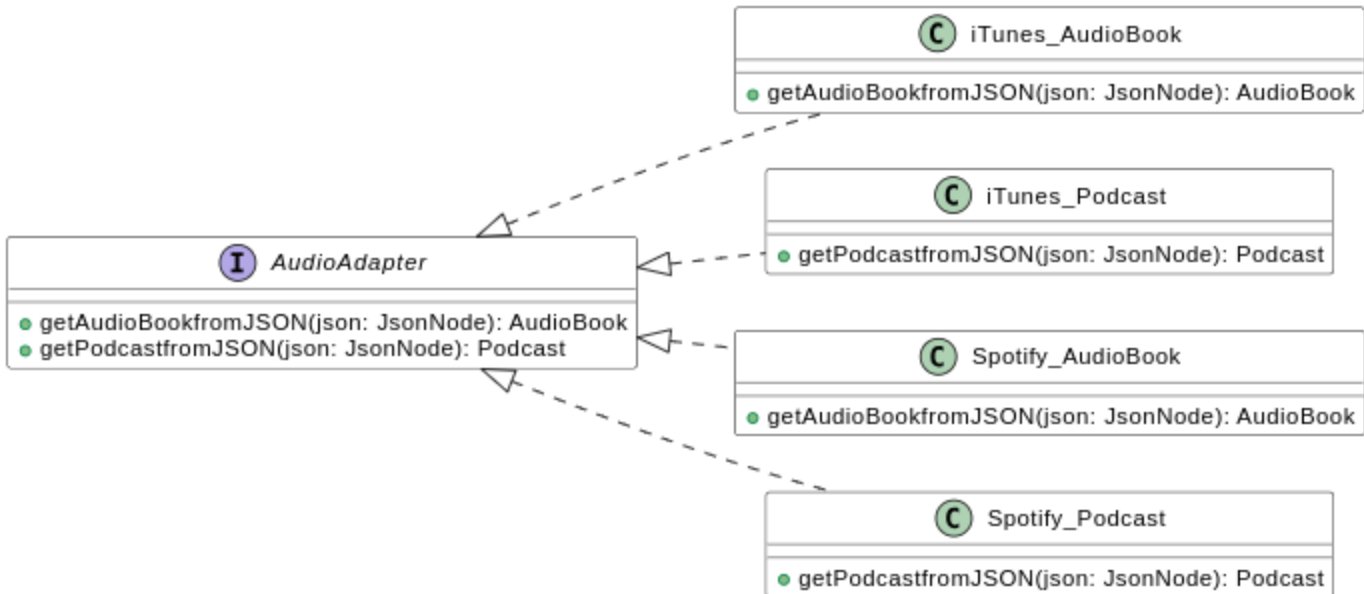
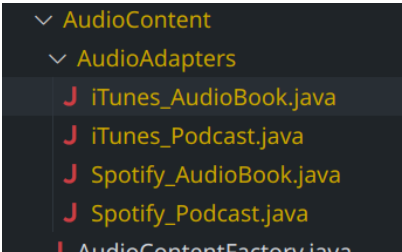
In addition to the Factory Pattern, we also utilize adapters to maintain a common data representation across the codebase. For instance, each service provider and content type has an adapter responsible for converting their respective objects into a common format. For example, an iTunes\_Podcast adapter converts an iTunes Podcast object into a standardized object format used throughout the application. This approach ensures consistency and ease of data handling across different service providers and content types.

```
public class iTunes_AudioBook extends AudioBook{
    public static final String ADAPTER_NAME = "iTunes_AudioBook";

    public AudioBook getAudioBookfromJSON(JsonNode json) {
        // Parse JSON and create AudioBook object
        String id = json.get(fieldName:"collectionId").getNumberValue().toString();
        String title = json.get(fieldName:"collectionName").getTextValue();
        String author = json.get(fieldName:"artistName").getTextValue();
        String description = json.get(fieldName:"description").getTextValue();
        String viewUrl = json.get(fieldName:"collectionViewUrl").getTextValue();

        AudioBook audioBook = new AudioBook();
        audioBook.setId(id);
        audioBook.setTitle(title);
        audioBook.setAuthor(author);
        audioBook.setDescription(description);
        audioBook.setUrlLink(viewUrl);

        System.out.println("Audio Book ID : " + id);
        return audioBook;
    }
}
```



## Implementation Details

### Front End

HTML and JS files to search and display audio books, podcasts and favorites, And perform the necessary requests to the server to carry out the functionality.

- audio.content.html
- audiobookcontent.view.html
- podcastcontent.view.html
- favourites.html
- Audio.js
- AudioBookContentView.js
- PodcastContentView.js
- Favourites.js

### Back End

- audiobookresource directory
  - AddAudioBook.java
  - FavouritesAudioBook.java
  - ListAudioBook.java
- podcastresource directory
  - AddPodcast.java
  - FavouritesPodcast.java
  - ListPodcast.java
- AudioContent directory in service

AudioContentFactory.java

iTunesAudioContent.java

SpotifyAudioContent.java

AudioAdapters directory

iTunes\_AudioBook.java

iTunes\_Podcast.java

Spotify\_AudioBook.java

Spotify\_Podcast.java

model

AudioBook.java

Podcast.java

UserAudioBook.java

UserPodcast.java

dao

UserAudioBookDao.java

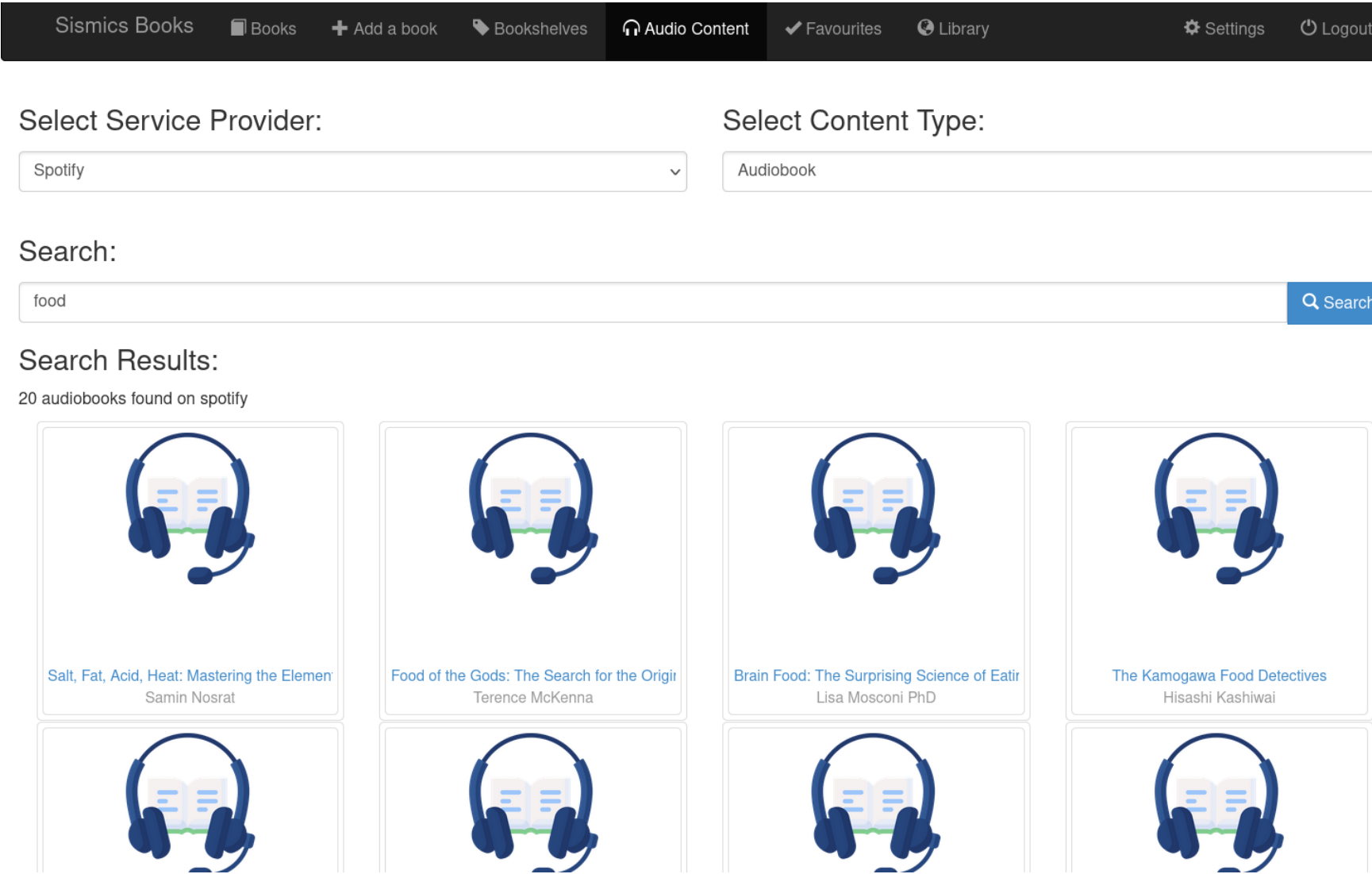
UserPodcastDao.java

AudioBookDao.java

PodcastDao.java

and corresponding `dto`s


## User Interface Pictures




# Salt, Fat, Acid, Heat: Mastering the Elements of Good Cooking

## Samin Nosrat

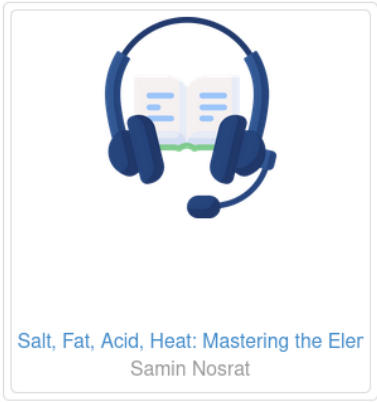
Author(s): Samin Nosrat Narrator(s): Samin Nosrat <b>Samin Nosrat reads "The Four Elements of Good Cooking," Part One of her <i>New York Times</i> bestseller <i>Salt, Fat, Acid, Heat: Mastering the Elements of Good Cooking</i></b><br><br>A visionary new master class in cooking that distills decades of professional experience into just four simple elements, from the woman declared "America's next great cooking teacher" by Alice Waters.<br><br>In the tradition of <i>The Joy of Cooking</i> and <i>How to Cook Everything</i> comes <i>Salt, Fat, Acid, Heat</i>, an ambitious new approach to cooking by a major new culinary voice. Chef and writer Samin Nosrat has taught everyone from professional chefs to middle school kids to author Michael Pollan to cook using her revolutionary, yet simple, philosophy. Master the use of just four elements&mdash;Salt, which enhances flavor; Fat, which delivers flavor and generates texture; Acid, which balances flavor; and Heat, which ultimately determines the texture of food&mdash;and anything you cook will be delicious. By explaining the hows and whys of good cooking, <i>Salt, Fat, Acid, Heat</i> will teach and inspire a new generation of cooks how to confidently make better decisions in the kitchen and cook delicious meals with any ingredients, anywhere, at any time.<br><br>Echoing Samin's own journey from culinary novice to award-winning chef, <i>Salt, Fat Acid, Heat</i> immediately bridges the gap between home and professional kitchens. With a lighthearted approach to kitchen science, Samin demystifies the four elements of good cooking for everyone.

 Add to Favourites

 Remove from Favourites

Audio BooksPodcasts

### Audio Book Favourites



Note - We have done the detailed documentation of changes in some of classes that can be referred to [here](#).