
Online Social Bookstore Application

Sprint Report

Team: Object Army

Philip Athanasopoulos Antyras 5113

Alexandros Pournaras 2528

1 Introduction

The following article is the report of the Software Engineering project of 2024. The source code and file files can also be found in [Github](#).

1.1 Purpose

The purpose of the project is to develop a social bookstore application. Other than the different functionalities of this application, the project should be designed in such a way that it is maintainable and extendable, as well fully documented. This shall result in fast and correct development and deployment of new features in the future. Another target of the project was to implement agile methods to develop software. In our case, we used **scrum**.

1.2 Document Structure

The rest of this document is structured as follows:

- [Section 2](#) specifies the technologies used during the development of the project.
- [Section 3](#) describes our Scrum team and specifies this Sprint's backlog.
- [Section 4](#) dives into the Use Cases, which derive from User Stories.
- [Section 5](#) explains the project's Architecture and Design.

2 Tools and Technologies



- **Java:** A high-level, class-based, object-oriented **programming language**. Widely used and famously quoted as “write once, run anywhere” as she has very few dependencies.
- **Spring Boot:** An open-source Java framework used for programming standalone, production-grade Spring-based applications.
- **Bootstrap:** An open-source **CSS framework** directed at responsive front-end web applications.
- **GitHub:** A **developer platform** that allows development teams to create, store and manage their code. It uses and extends the Git software.
- **JUnit:** A **unit testing framework** for java.
- **PostgreSQL:** A free and open-source **relational database** management system (RDBMS) emphasizing extensibility and **SQL** compliance.
- **Jira:** A product development tool that allows bug tracking, issue tracking and agile project management.
- **IntelliJ:** An IDE for developing computer software written in Java and other JVM-based languages.
- **Maven:** A **build automation** tool used for, among others, java projects.

3 Scrum team and Sprint Backlog

3.1 Scrum team

Product Owner	Alex Pournaras
Scrum Master	Philip Athanasopoulos
Development Team	Philip Athanasopoulos, Alex Pournaras

3.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories	Description
1	9/3/2024	22/3/2024	2	None	Determine the tools and technologies to be used to build the project. Create use cases, domain classes and UML diagrams.
2	24/3/2024	07/04/2024	2	US1, US2, US3	Implement basic user functionalities such as creating and logging in/out of an account.
3	07/04/2024	21/04/2024	2	US5, US6, US7, US8, US9	Complete book offer functionalities: Add, remove book offers with info for them. Browse book request list. Browse interested users list for each book. Select a user to hand each book

4	24/04/2024	05/05/2024	2	US4, US10, US11	Add User Profile editing and book search functionalities
5	06/05/2024	19/05/2024	2	None	Create separate database for testing and test each class.

4 Use Cases

4.1 Create account

Use case ID	UC1
Actors	User
Preconditions	The user has entered the website
Main flow of events	<ol style="list-style-type: none"> 1. The use case starts when the user clicks on the “create an account” button. 2. The user is asked to enter the information needed to create a new account. 3. The user presses on the “create my account” button.
Post conditions	4. A new account for the user has been created with an empty profile.
Alternative flow 1	<ol style="list-style-type: none"> 1. The user enters an email that is already registered by an account. 2. The user is asked to enter another email.
Post conditions	An error message is displayed to the user
Alternative flow 2	<ol style="list-style-type: none"> 1. The field entered by the user is not in correct form. 2. The user is asked to re-enter the field.
Post conditions	An error message is displayed to the user

Login

Use case ID	UC2
Actors	User
Pre conditions	The user has entered the website and has an account.
Main flow of events	5. The use case starts when the user clicks on the “login” button. 6. The user is asked to enter the information needed to login. 7. The user presses on the “log me in” button.
Post conditions	8. The user has been logged in to their account.
Alternative flow 1	9. The user enters an email that is not registered.
Post conditions	10. The system pops an error message (email not registered) and the user is asked to fill the login form again.
Alternative flow 2	11. The field enters the wrong password for the given email account.
Post conditions	12. The system pops an error message (email-password doesn’t match) and the user is asked to fill the login form again.

Logout

Use case ID	UC3
Actors	User
Pre conditions	The user has entered the website and has logged into their account.
Main flow of events	13. The use case starts when the user clicks on the “logout” button. 14. The user is asked whether they really want to log out. 15. The user presses on the “yes” button. 16. The user is logged out of their account and brought to the main screen. of the application.
Post conditions	17. The user has been logged in to their account.

Alternative flow 1	18. The user presses on the “no” button.
Post conditions	19. The user remains logged in

Edit profile

Use case ID	UC4
Actors	User
Pre conditions	The user has entered the website and has logged into their account.
Main flow of events	20. The use case starts when the user clicks on the “edit profile” button. 21. The user is brought to the profile editing section. 22. The user edits any field they want to change. 23. The user presses on the “save changes” button.
Post conditions	24. The users’ profile has been updated appropriately.
Alternative flow 1	25. The user presses the “cancel” button. 26. The user is asked whether they want to discard the changes. 27. The user presses on the “yes” button. 28. The profile fields remain the same.
Post conditions	29. The users profile remains the same.
Alternative flow 2	30. The user presses the “cancel” button. 31. The user is asked whether they want to discard the changes. 32. The user presses on the “no” button. 33. The user continues the editing of the profile.
Post conditions	34. The user stays in the profile editing section and the fields entered by them are still in the editing section.

Add book offer

Use case ID	UC5
Actors	User
Pre conditions	The user has entered the website and has logged into their account.
Main flow of events	35. The use case starts when the user clicks on the “add book offer” button. 36. The user fills in the required book description fields. 37. The user clicks on the “post book offer” button.
Post conditions	38. A new book offer has been posted on the site.
Alternative flow 1	39. The user enters inadequate/incompatible fields. 40. The user gets notified that some fields need to be edited.
Post conditions	41. The system pops an error message (inadequate/incompatible fields) and the user is asked to fill the book offer form again.
Alternative flow 2	42. The user clicks on the “cancel book offer” button. 43. The user is asked whether they want to discard the book offer. 43.1. The user clicks on the “no” button. 43.1.1. The user continues with their book offer posting. 43.2. The user presses on the “yes” button. 43.2.1. The book offer posting is discarded.
Post conditions	44. The book offer posting is discarded.

Browse book request list

Use case ID	UC6
Actors	User
Pre conditions	The user has logged into their account and they have ≥ 1 book for offer.
Main flow of events	45. The use case starts when the user clicks on the “my offers” button. 46. The user selects the desired book from their book offer list.

	47. The requests for the desired book are listed.
Post conditions	48. A new book offer has been posted on the site.
Alternative flow 1	49. The user selects the desired book from their book offer list. 50. The requests for the desired book are listed. 51. The user clicks on the “go back” button.
Post conditions	52. The user is brought back to the main page

Select interested user

Use case ID	UC7
Actors	User
Pre conditions	The user is browsing their request list and they have ≥ 1 requests.
Main flow of events	53. The use case starts when the user clicks on the “my offers” button. 54. The user then selects the requestee. 55. The user clicks on the “hand book” button. 56. The user is asked to commit their choice. 57. The user clicks on the “yes, I’m sure”. 58. The system notifies the selected user that they have been handed the book. 59. The system notifies every other user in the list that the book has been handed elsewhere 60. That book is discarded from the book offer list.
Post conditions	61. That book has been handed to the selected requestee.
Alternative flow 1	62. The user then selects the requestee. 63. The user clicks on the “handbook” button. 64. The user is asked to commit their choice. 65. The user clicks on the “Cancel” button.
Post conditions	66. The user is brought back to the request list.

Access contact info

Use case ID	UC8
Actors	User
Pre conditions	The user is browsing their request list and they have ≥ 1 requests.
Main flow of events	67. The use case starts when the user clicks on the “my offers” button. 68. The user then selects the requestee. 69. The user clicks on the “hand book” button. 70. The user is asked to commit their choice. 71. The user clicks on the “yes, I’m sure”. 72. The system notifies the selected user that they have been handed the book. 73. The system notifies every other user in the list that the book has been handed elsewhere 74. That book is discarded from the book offer list.
Post conditions	75. That book has been handed to the selected requestee.
Alternative flow 1	76. The user then selects the requestee. 77. The user clicks on the “hand book” button. 78. The user is asked to commit their choice. 79. The user clicks on the “Cancel” button.
Post conditions	80. The user is brought back to the request list.

Remove book offer

Use case ID	UC9
Actors	User

Pre conditions	The user is browsing their book offer list.
Main flow of events	81. The use case starts when the user clicks on the “my offers” button. 82. The user selects a book offer they want to remove. 83. The user clicks on the “remove offer” button. 84. The user is asked whether they want to remove the offer. 85. The user clicks on the “yes” button. 86. That book is removed from the book offer list. 87. The system removes the book from every requestee book request list.
Post conditions	88. That book offer has been removed
Alternative flow 1	89. The user selects a book offer they want to remove. 90. The user clicks on the “remove offer” button. 91. The user is asked whether they want to remove the offer. 92. The user clicks on the “no” button.
Post conditions	93. The user is brought back to the book offer panel

Search for books

Use case ID	UC10
Actors	User
Pre conditions	The user has logged in.
Main flow of events	94. The use case starts when the user clicks on the “search” button. 95. The user types the name/author/category of the book they want to search for. 96. The user presses on the “go” button. 97. The system finds appropriate results. 98. The search results are displayed to the user.

Post conditions	99. The search results are displayed to the user.
Alternative flow 1	100. The use case starts when the user clicks on the “search” button. 101. The user types the name/author/category of the book they want to search for. 102. The user presses on the “Go” button. 103. The system can find no matches for the given query.
Post conditions	104. The system pops a message to the user (no books were found).

Search for exact book

Use case ID	UC10
Actors	User
Pre conditions	The user has logged in.
Main flow of events	105. The use case starts when the user clicks on the “search” button. 106. The user selects the “exact” search option. 107. The user types the name/author/category of the book they want to search for. 108. The user presses on the “go” button. 109. The system finds the book. 110. The search result is displayed to the user.
Post conditions	111. The search result is displayed to the user.
Alternative flow 1	112. The use case starts when the user clicks on the “search” button. 113. The user types the name/author/category of the book they want to

	search for. 114. The user presses on the “Go” button. 115. The system can find no matches for the given query.
Post conditions	116. The system pops a message to the user (book wasn’t found.)

Search for approximate books

Use case ID	UC11
Actors	User
Pre conditions	The user has logged in.
Main flow of events	117. The use case starts when the user clicks on the “search” button. 118. The user selects the “approximate” search option. 119. The user types the name/author/category of the book they want to search for. 120. The user presses on the “Go” button. 121. The system can find some matches for the given query.
Post conditions	122. The system displays the search results to the user.
Alternative flow 1	123. The use case starts when the user clicks on the “search” button. 124. The user types the name/author/category of the book they want to search for. 125. The user presses on the “Go” button. 126. The system can find no matches for the given query.
Post conditions	127. The system pops a message to the user (book wasn’t found.)

Browse recommended books

Use case ID	UC12
--------------------	------

Actors	User
Pre conditions	The user has logged in.
Main flow of events	128. The use case starts when the user clicks on the “browse recommended books” button. 129. The user is brought to the recommended books section. 130. Recommended books are displayed to the user based on their profile.
Post conditions	131. The system displays to the user their recommended books list.

Select search result

Use case ID	UC13
Actors	User
Pre conditions	The user is displayed results from the search/recommended books sections.
Main flow of events	132. The user clicks on a book offer result.
Post conditions	133. The system displays to the user the selected book’s offer description and options.

Request book

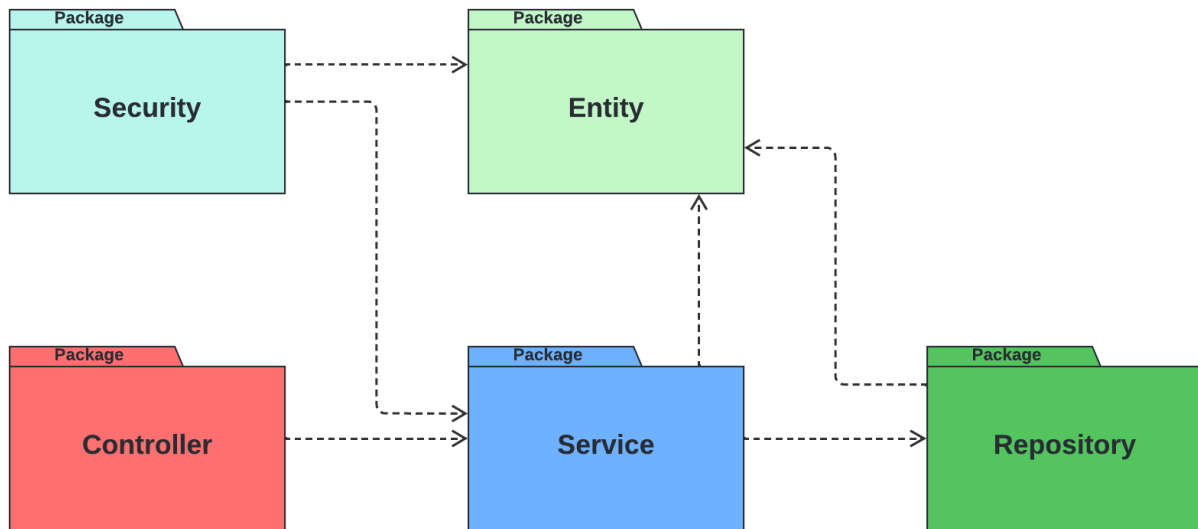
Use case ID	UC14
Actors	User
Pre conditions	The user has selected a book offer.
Main flow of events	134. The use case starts when the user clicks on a book offer. 135. The user clicks on the “Request” button. 136. The system notifies the offeror for the request. 137. The system adds the book to the “requested” list of the user.
Post	138. A request for the book has been submitted.

5 Design

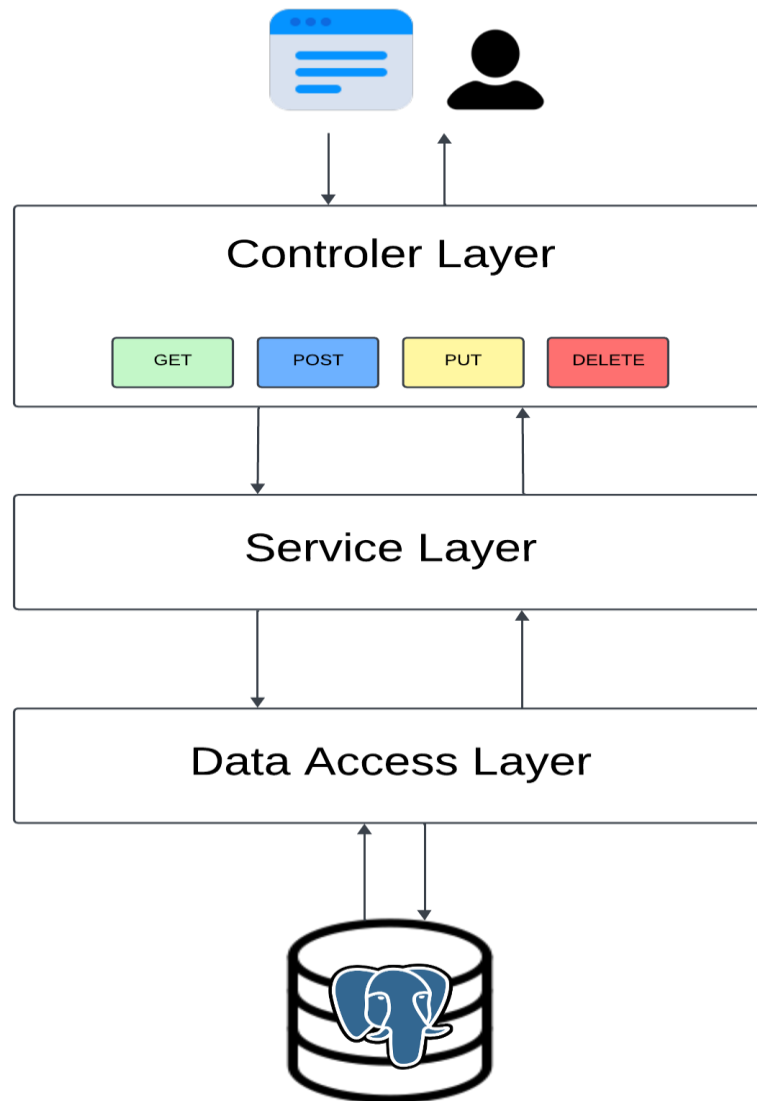
5.1 Architecture

The project consists of **5 packages**.

1. **Entity** : The entity package contains the domain classes of the project. Those classes represent real world objects (user, book, etc.).
2. **Repository**: The repository package contains the classes that extend the **JpaRepository** Interface. They allow Spring to map Entity classes into SQL tables. They also provide query methods to save, delete and update java object data into the SQL schema of the project.
3. **Service**: The Service package contains classes responsible for transferring between the backend and the front-end. The classes apply domain logic to determine whether the incoming data requests are valid.
4. **Controller**: The controller package contains classes that take care of **HTTP requests** made on the project's front-end. Each class is responsible for calling the appropriate Service class based on incoming **CRUD** operations and loading the view of each endpoint if needed.
5. **Security**: The security package contains the configurations needed to securely authorize user access. It determines which endpoints should be public and/or accessible for a specific type of user.



The project uses the **3-Tier architecture** which means that the controller requests data from the service layer. The service layer determines if the controller should have access to the requested data given the user credentials. Finally, the service queries the data from the repository and sends it back to the controller.



5.2 Design

The **Entity** package consists of 4 subpackages:

1. Book
2. Recommend
3. Search
4. User

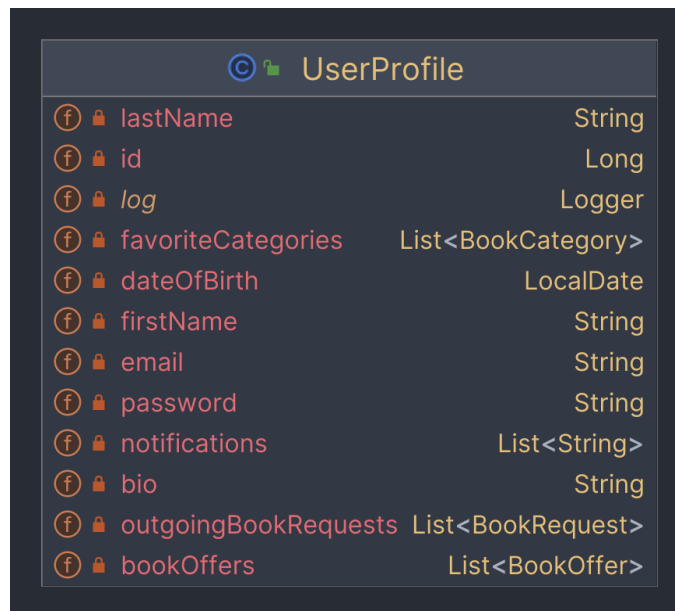


The **Book** packages contains domain classes associated with books of the bookstore. These classes are:

- Book
- Book Category
- Book Offer
- Book Request
- Author

All those classes are JPA Entities

The **User** package contains the user class which contains all the users' fields. Other than the user's common data (name, email, etc.) the user also holds a list of the books they offer, a list of books they have requested and a notifications list. The user class implements the **UserDetails** interface to make Spring Security authentication possible. It's important to note that the password field is hashed and can only be extracted by the security package.

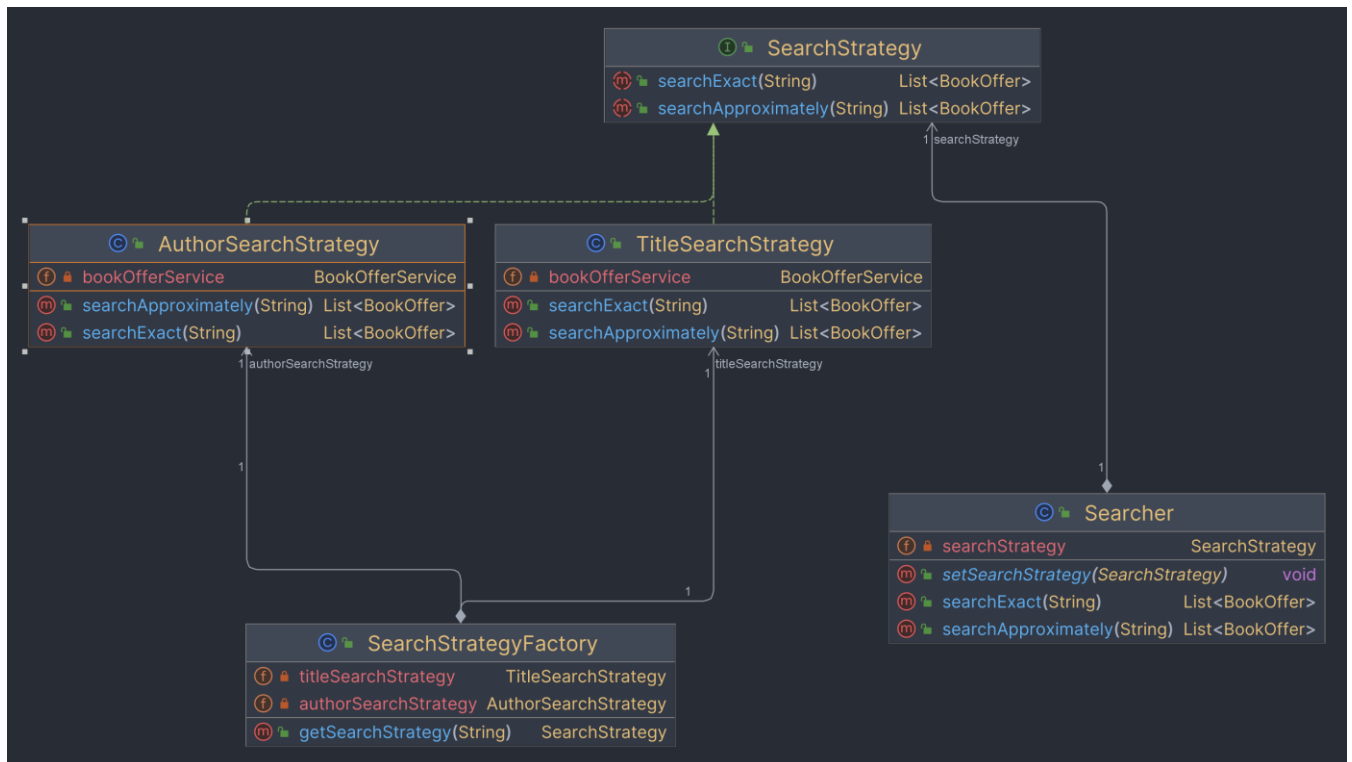


© UserProfile		
Ⓜ 🔒	lastName	String
Ⓜ 🔒	id	Long
Ⓜ 🔒	log	Logger
Ⓜ 🔒	favoriteCategories	List<BookCategory>
Ⓜ 🔒	dateOfBirth	LocalDate
Ⓜ 🔒	firstName	String
Ⓜ 🔒	email	String
Ⓜ 🔒	password	String
Ⓜ 🔒	notifications	List<String>
Ⓜ 🔒	bio	String
Ⓜ 🔒	outgoingBookRequests	List<BookRequest>
Ⓜ 🔒	bookOffers	List<BookOffer>

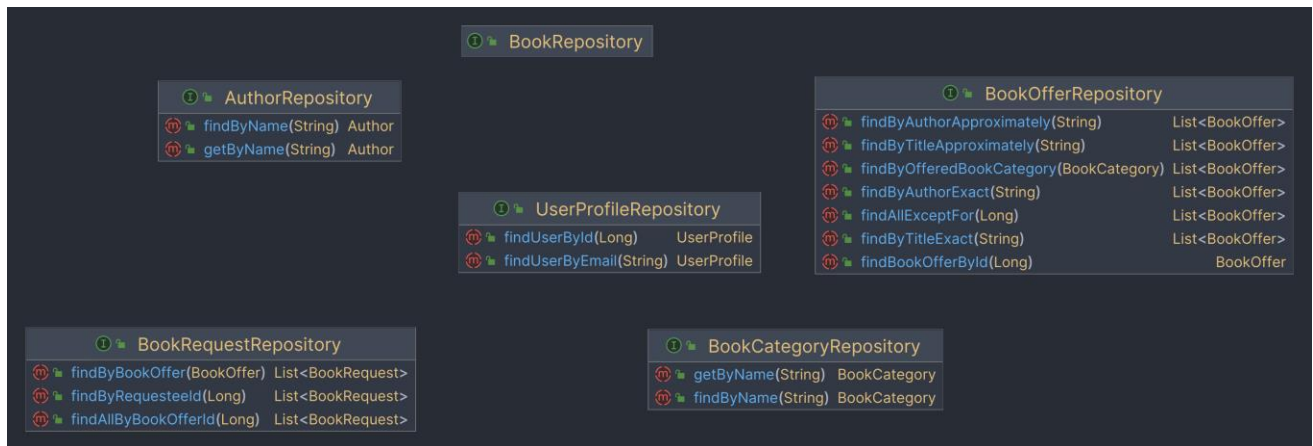
The Recommend package contains classes for the purpose of recommending books to the user. To do so, the **Strategy Pattern** is used. The BookOfferRecommender, which is the **context class**, has a Book offer recommend Strategy object. This is an interface that can be implemented by child classes to recommend books in a specific way. An example of this implementation is the CategoryBasedRecommendStrategy.



The Search package contains classes for the purpose of searching for books the user is looking for. Here the **Strategy Pattern** is also used, with **Searcher** as the context class and **SearchStrategy** as the interface. The interface has 2 separate methods, one for exact searching and the other for approximate searching. There is an implementation for Title based and Author based searching.



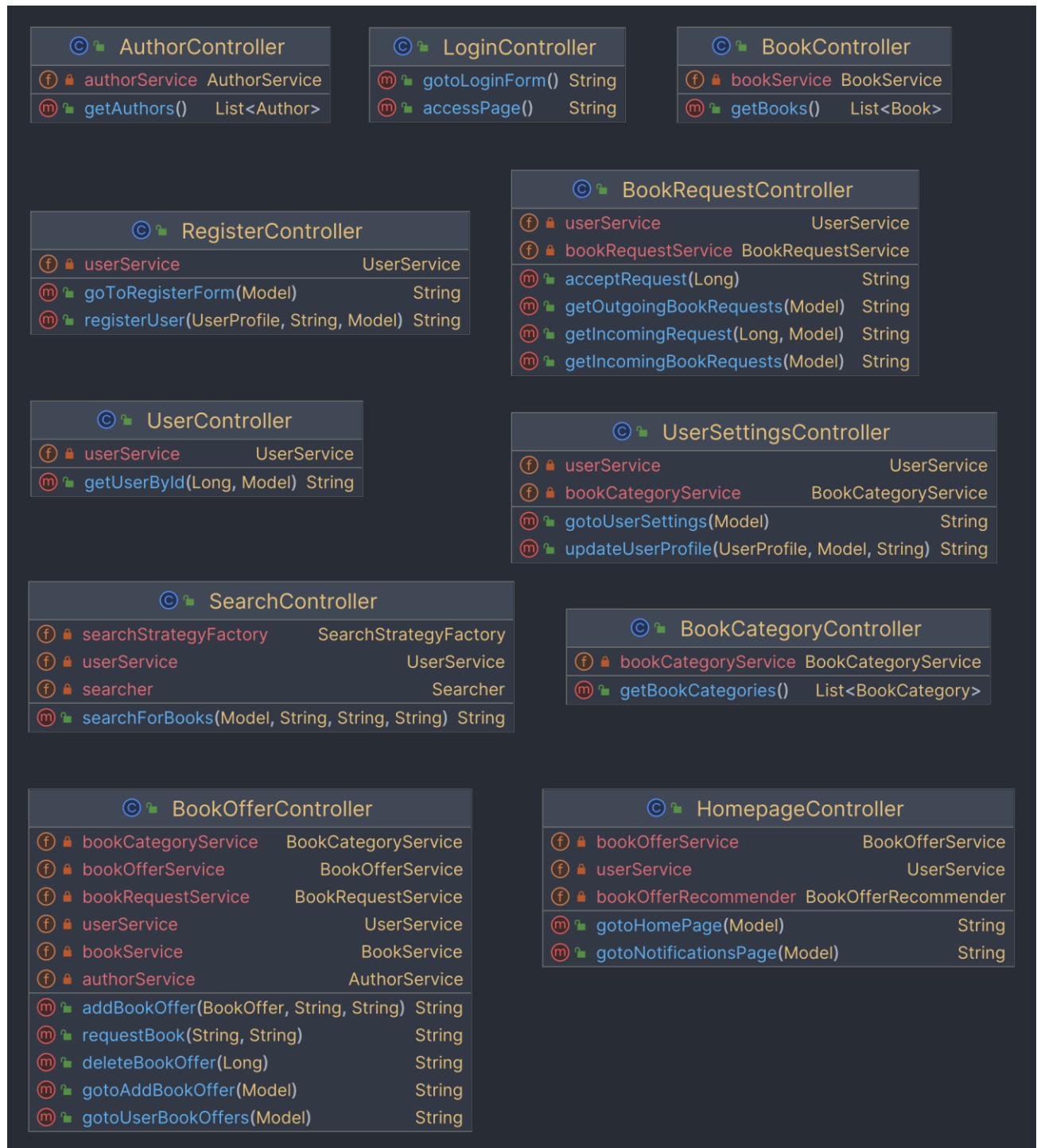
The **Repository package** contains interfaces that extend the **JpaRepository** interface. These interfaces are used to map the classes of the Entity package into SQL tables. Each class has query methods to read/write data to the corresponding SQL table.



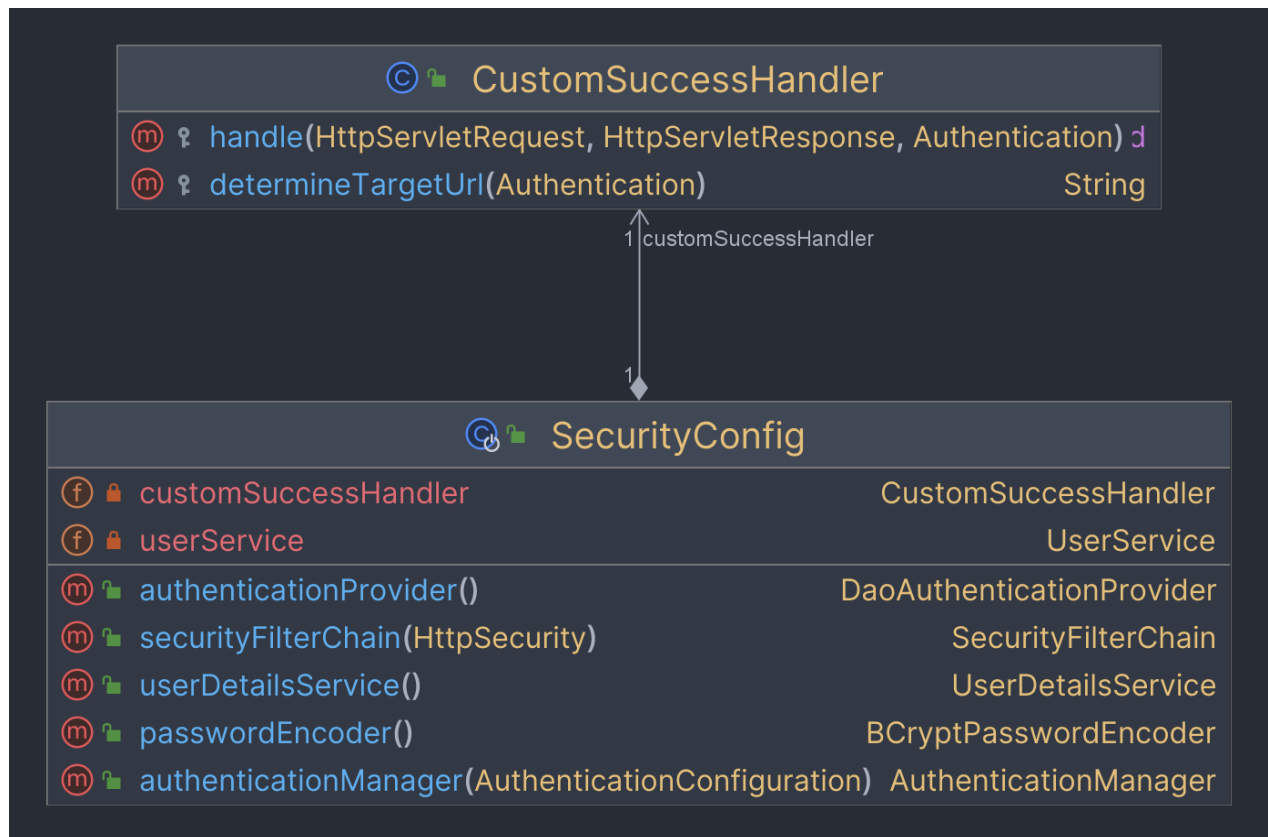
The **Service package** is responsible for transferring data between the database and the frontend. Each service class uses the corresponding interfaces method to perform read/write operations. On top of that the service applies the rules under which the read/write operations can be performed, i.e. the business logic of the project.



The **Controller package** contains classes responsible for responding to HTTP requests made by the user to the applications endpoints. Each controller can respond to HTTP post and delete methods using the required service class from the service package. The controllers are also responsible for displaying the views (html files) so that the user can navigate the application.



The **Security package** is responsible for managing which users have access to the application's different endpoints. Its config class determines that unauthenticated users can only access the login and sign-up endpoints. Furthermore, it uses the user service to load **userProfile** objects and authenticate them using the **BCryptPasswordEncoder**.



Class Name: Book	
Responsibilities: <ul style="list-style-type: none"> Stores details about books (title, author etc) Manage the list of authors associated with the book. Manage the list of categories associated with the book. 	Collaborations: <ul style="list-style-type: none"> Author - associated with the authors who wrote the book. BookCategory - associated with the categories the book belongs to.

Class Name: Author	
Responsibilities: <ul style="list-style-type: none"> ▪ Store information about an author, including their unique identifier and name 	Collaborations: <ul style="list-style-type: none"> ▪ Book - likely to be associated with books written by the author.

Class Name: BookCategory	
Responsibilities: <ul style="list-style-type: none"> ▪ Store information about a book category, including its unique identifier and name. 	Collaborations: <ul style="list-style-type: none"> ▪ Book - associated with books that belong to this category.

Class Name: BookOffer	
Responsibilities: <ul style="list-style-type: none"> ▪ Store information about a book offer, including its unique identifier, the offeror, the offered book, a 	Collaborations: <ul style="list-style-type: none"> ▪ UserProfile - associated with the user making the offer.

<p>description, requests, and the post date.</p> <ul style="list-style-type: none"> Manage the list of requests associated with the book offer. 	<ul style="list-style-type: none"> Book - the book being offered. BookRequest - requests made by other users for the offered book.
--	--

Class Name: BookRequest	
Responsibilities: <ul style="list-style-type: none"> Store information about a book request, including its unique identifier, the requestee, and the associated book offer. 	Collaborations: <ul style="list-style-type: none"> UserProfile - the user making the request. BookOffer - the book offer being requested.

Class Name: AuthorService	
Responsibilities: <ul style="list-style-type: none"> Retrieve a list of all authors from the repository. Retrieve an author by name. Retrieve an author by ID. Add a new author to the repository. Retrieve an author by name or create a new one if not found. Save an author to the repository if the author does not already exist. 	Collaborations: <ul style="list-style-type: none"> Author - the entity being managed by the service. AuthorRepository - interacts with the repository to perform CRUD operations on the Author entity.

Class Name: BookCategoryService	
Responsibilities: <ul style="list-style-type: none"> Retrieve a list of all book categories from the repository. Add a new book category to the repository. 	Collaborations: <ul style="list-style-type: none"> BookCategoryRepository - interacts with the repository to perform CRUD operations on the BookCategory entity. BookCategory - the entity being

<ul style="list-style-type: none"> ▪ Find a book category by name. ▪ Retrieve a book category by name. ▪ Retrieve a book category by name or create a new one if not found. ▪ Save a book category to the repository if it does not already exist. 	managed by the service.
--	-------------------------

Class Name: BookOfferService	
Responsibilities: <ul style="list-style-type: none"> ▪ Save a book offer to the repository, ensuring associated book and user information is also saved. ▪ Retrieve a list of all book offers from the repository. ▪ Retrieve a book offer by its ID. ▪ Delete a book offer by its ID, including associated book requests. ▪ Perform searches for book offers by title and author, both approximate and exact matches. ▪ Retrieve book offers by category. ▪ Retrieve all book offers except for a specific one. 	Collaborations: <ul style="list-style-type: none"> ▪ BookOfferRepository - interacts with the repository to perform CRUD operations on the BookOffer entity. ▪ BookService - manages book-related operations, such as saving books associated with offers. ▪ BookRequestService - manages book request-related operations, such as deleting requests associated with deleted offers. ▪ UserProfile - the entity representing the offeror of the book offer. ▪ BookCategory - the entity representing the category associated with the book offer. ▪ BookOffer - the entity being managed by the service.

Class Name: BookRequestService	
Responsibilities: <ul style="list-style-type: none"> ▪ Save a book request to the repository, ensuring associated book offer and user information is also saved. ▪ Delete a book request from the repository, including updating associated user profiles and book offers. 	Collaborations: <ul style="list-style-type: none"> ▪ BookRequestRepository - interacts with the repository to perform CRUD operations on the BookRequest entity. ▪ UserService - manages user-related operations, such as retrieving user profiles by ID. ▪ BookOfferService - manages book offer-

<ul style="list-style-type: none"> ▪ Retrieve a book request by its ID. ▪ Delete a book request by its ID. ▪ Accept a book request, notifying the requesting user and declining other requests for the same book offer. ▪ Delete all book requests associated with a specific book offer. 	<p>related operations, such as retrieving book offers by ID.</p> <ul style="list-style-type: none"> ▪ UserProfile - the entity representing the user making the request. ▪ BookOffer - the entity representing the book offer for which the request is made. ▪ BookRequest - the entity being managed by the service.
---	--

Class Name: BookService	
Responsibilities: <ul style="list-style-type: none"> ▪ Retrieve a list of all books from the repository. ▪ Save a book to the repository, ensuring associated authors and categories are also saved. 	Collaborations: <ul style="list-style-type: none"> ▪ BookRepository - interacts with the repository to perform CRUD operations on the Book entity. ▪ AuthorService - manages author-related operations, such as saving authors associated with the book. ▪ BookCategoryService - manages book category-related operations, such as saving categories associated with the book. ▪ Book - the entity representing the book being managed by the service. ▪ Author - the entity representing the authors of the book. ▪ BookCategory - the entity representing the categories of the book.

Class Name: UserService	
Responsibilities: <ul style="list-style-type: none"> ▪ Save a user profile to the repository, ensuring the password is hashed before saving. ▪ Delete a user profile from the repository. ▪ Load a user profile by username (email) for Spring Security 	Collaborations: <ul style="list-style-type: none"> ▪ UserProfileRepository - interacts with the repository to perform CRUD operations on the UserProfile entity. ▪ BCryptPasswordEncoder - used to encode passwords before saving them to the repository. ▪ UserProfile - the entity representing the

<p>authentication.</p> <ul style="list-style-type: none">▪ Retrieve a user profile by its ID.▪ Update user profile details, including first name, last name, email, date of birth, password, bio, and favorite categories.	<p>user profile being managed by the service.</p> <ul style="list-style-type: none">▪ UserDetails - the interface representing a Spring Security user for authentication.
---	---