Online Social Bookstore Application

Sprint Report

Vasilis Nikas 3143

Dimitris Moschos 2301

Thanasis Tsiggenopoulos 4189

VERSIONS HISTORY

Date	Version	Description	Author
20/5/2024	1.0	Final Report	Team

1 Introduction

This document provides information concerning the **final** sprint of the project.

1.1 Purpose

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies this Sprint's backlog. Section 3 contains the use cases of this project. Finally, Section 4 specifies the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Team members
Scrum Master	Team members
Development Team	Team members

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**25/3/2024**	**5/4/2024**	**2**	**US1, US2, US3, US4**
2	**6/4/2024**	**20/4/2024**	**2**	**US5, US9**
3	**21/4/2024**	**5/5/2024**	**2**	**US6, US7, US8**
4	**6/5/2024**	**15/5/2024**	**2**	**US10**

3 Use Cases

3.1 <Use Case 1>

Use case ID	UC1
Actors	User
Pre conditions	-
Main flow of events	1. The use case starts when the user opens the app to a browser. 2. The user chooses the option register. 3. The user inserts the username and the password presses the Register button. 4. The system creates the user.
Alternative flow	If the user inserts an already existing username then the system shows a warning message.
Post conditions	The system created the user account.

3.2 <Use Case 2>

Use case ID	UC2
Actors	User
Pre conditions	The user must already create an account.
Main flow of events	1. The use case starts when the user chooses the Login option. 2. The user inserts the username and the password and presses the Login button. 3. The system login the user to the main app.
Alternative flow	If the user inserts a wrong username then the system shows a warning message.
Post conditions	The user login to his/her account.

3.3 <Use Case 3>

Use case ID	UC3
Actors	User
Pre conditions	The user must be logged in to the account.
Main flow of events	1. The use case starts when the user clicks the Logout button. 2. The system logs out the user.
Alternative flow	-
Post conditions	The user is no longer logged in his/her account.

3.4 <Use Case 4>

Use case ID	UC4
Actors	User
Pre conditions	The user must already create an account and login.
Main flow of events	1. The use case starts when the user clicks the warning message at the home page. 2. The system returns a form for profile creation. 3. The user inserts the proper information that the system asks for and presses the Save Profile button. 4. The system save the user profile.
Alternative flow 1	If the user doesn't add the correct information may has problem later with the app usage.
Post conditions	The user profile is created.

3.5 <Use Case 5>

Use case ID	UC5
Actors	User
Pre conditions	The user must sign in to the app.
Main flow of events	1. The use case starts when the user presses the My Books option and then the Create Book Offer option. 2. The system returns a form for book creation. 3. The user inserts the proper information and presses the Create button. 4. The system saves the book offer to the database.
Alternative flow 1	If the user presses the Go Back option, then the system returns the user to My Books view.
Post conditions	The system creates the book offer.

3.6 <Use Case 6>

Use case ID	UC6
Actors	User
Pre conditions	The user must create at least one book offer.
Main flow of events	1. The use case starts when the user, at My Books view, selects a book and presses the Requests button at the Actions section. 2. The system returns the users that have requested the specified book.
Alternative flow 1	-
Post conditions	The system informs the user for the requesting users.

3.7 <Use Case 7>

Use case ID	UC7
Actors	User
Pre conditions	The user must create at least one book offer and at least one user must request the specified book.
Main flow of events	1. The use case starts when the user, at the requesting users view, selects a user by pressing the Accept Request at the right of the selected user. 2. The system informs the selected user that he/she can take the book and the rest of the users that they cannot.
Alternative flow 1	-
Post conditions	The user chooses a user to give a book.

3.8 <Use Case 8>

Use case ID	UC8
Actors	User
Pre conditions	The user must create at least one book offer and at least one user must request the specified book.
Main flow of events	1. The use case starts when the user presses the username of a user at the Requests for this book view. 2. The system returns the page with info for the selected user.
Alternative flow 1	-
Post conditions	The user accesses the info so he/she can contact the other user.

3.9 <Use Case 9>

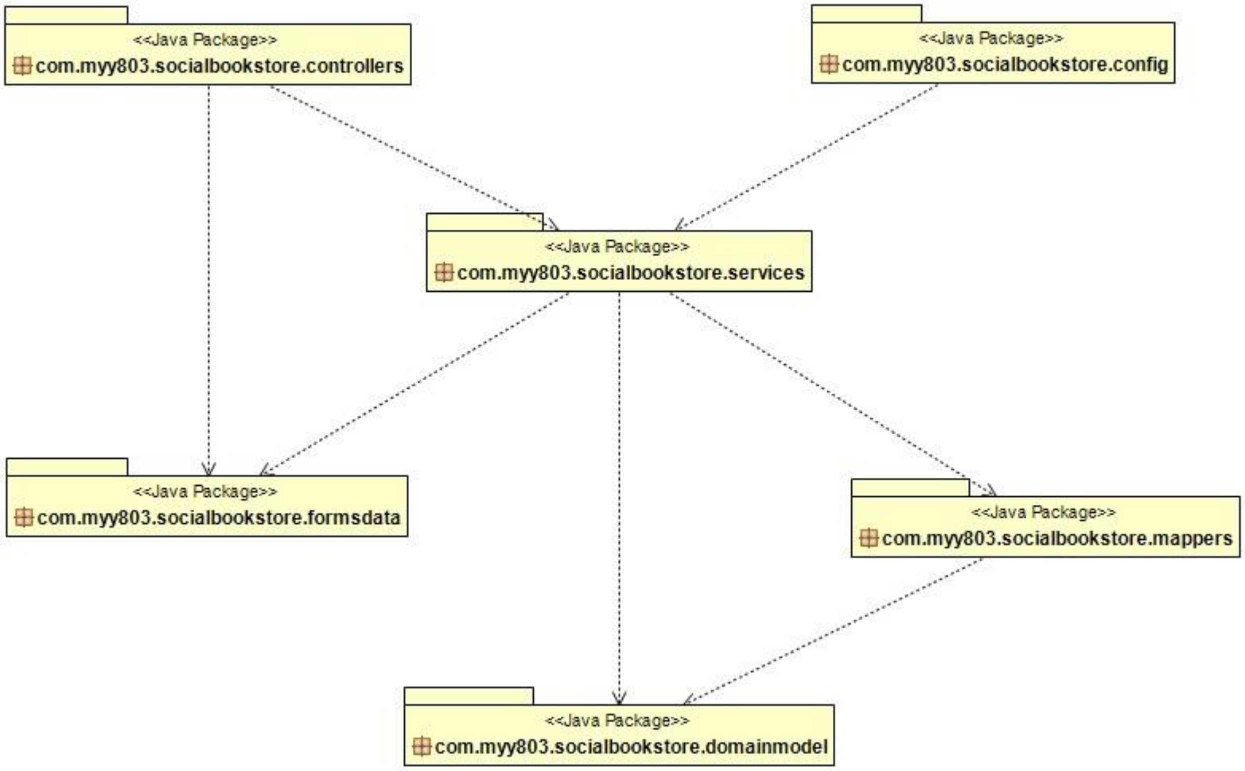
Use case ID	UC9
Actors	User
Pre conditions	The user must create at least one book offer.
Main flow of events	1. The use case starts when the user chooses the option Delete at my Books for a specific book. 2. The system deletes the chosen book and refresh My Books view.
Alternative flow 1	-
Post conditions	The system deletes the chosen book.

3.10 <Use Case 10>

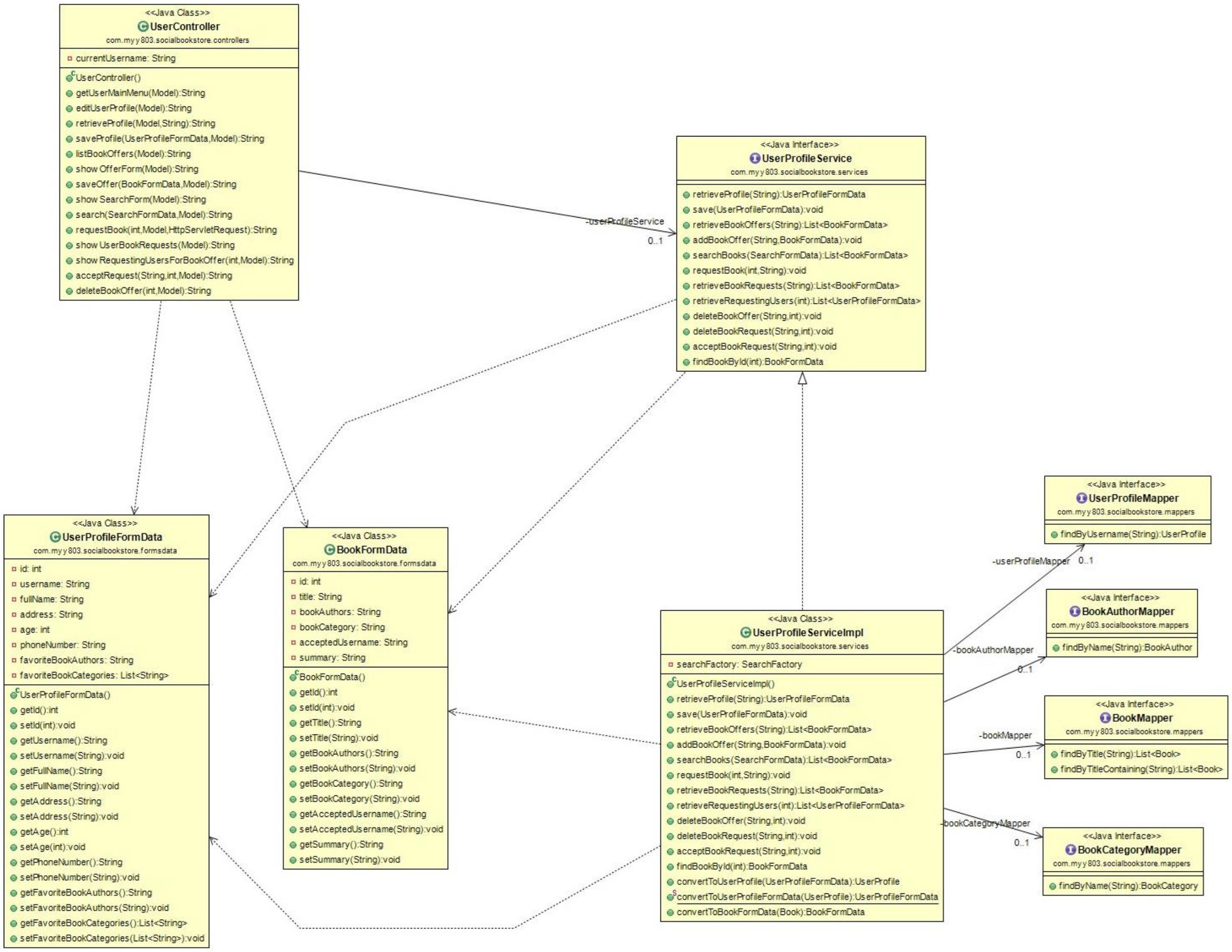
Use case ID	UC10
Actors	User
Pre conditions	The user must sign in to his/her account.
Main flow of events	1. The use case starts when the user presses the Search option at the menu. 2. The system returns a search form. 3. The user inserts the proper info and selects Search. 4. The system searches for the books that match with the search criteria and return the books in a list. 5. The user can Add request for a book that has interest in.
Alternative flow 1	-
Post conditions	The system brings the wanted books for the user to request for.

4 Design

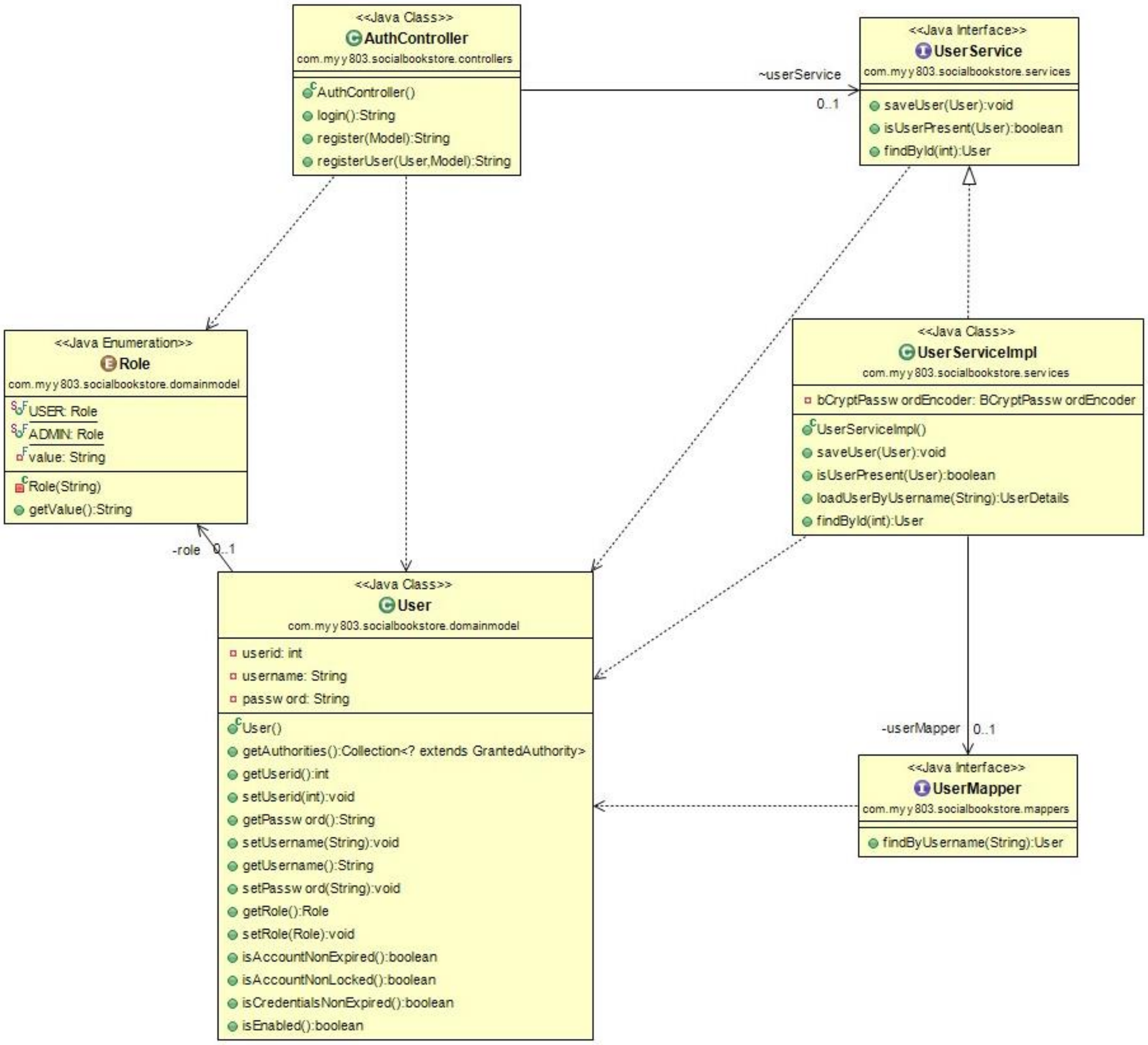
4.1 Architecture


<<Java Package>>
com.myy803.socialbookstore.controllers
<<Java Package>>
com.myy803.socialbookstore.config
<<Java Package>>
com.myy803.socialbookstore.services
<<Java Package>>
com.myy803.socialbookstore.formsdata
<<Java Package>>
com.myy803.socialbookstore.mappers
<<Java Package>>
com.myy803.socialbookstore.domainmodel

4.2 Design



<<Java Class>>
UserController
com.myy803.socialbookstore.controllers
currentUsername: String
UserController()
getUserMainMenu(Model):String
editUserProfile(Model):String
retrieveProfile(Model,String):String
saveProfile(UserProfileFormData,Model):String
listBookOffers(Model):String
showOfferForm(Model):String
saveOffer(BookFormData,Model):String
showSearchForm(Model):String
search(SearchFormData,Model):String
requestBook(int,Model,HttpServletRequest):String
showUserBookRequests(Model):String
showRequestingUsersForBookOffer(int,Model):String
acceptRequest(String,int,Model):String
deleteBookOffer(int,Model):String
-userProfileService
0..1
<<Java Interface>>
UserProfileService
com.myy803.socialbookstore.services
retrieveProfile(String):UserProfileFormData
save(UserProfileFormData):void
retrieveBookOffers(String):List<BookFormData>
addBookOffer(String,BookFormData):void
searchBooks(SearchFormData):List<BookFormData>
requestBook(int,String):void
retrieveBookRequests(String):List<BookFormData>
retrieveRequestingUsers(int):List<UserProfileFormData>
deleteBookOffer(String,int):void
deleteBookRequest(String,int):void
acceptBookRequest(String,int):void
findBookById(int):BookFormData
<<Java Class>>
UserProfileFormData
com.myy803.socialbookstore.formsdata
id: int
username: String
fullName: String
address: String
age: int
phoneNumber: String
favoriteBookAuthors: String
favoriteBookCategories: List<String>
UserProfileFormData()
getId():int
setId(int):void
getUsername():String
setUsername(String):void
getFullName():String
setFullName(String):void
getAddress():String
setAddress(String):void
getAge():int
setAge(int):void
getPhoneNumber():String
setPhoneNumber(String):void
getFavoriteBookAuthors():String
setFavoriteBookAuthors(String):void
getFavoriteBookCategories():List<String>
setFavoriteBookCategories(List<String>):void
<<Java Class>>
BookFormData
com.myy803.socialbookstore.formsdata
id: int
title: String
bookAuthors: String
bookCategory: String
acceptedUsername: String
summary: String
BookFormData()
getId():int
setId(int):void
getTitle():String
setTitle(String):void
getBookAuthors():String
setBookAuthors(String):void
getBookCategory():String
setBookCategory(String):void
getAcceptedUsername():String
setAcceptedUsername(String):void
getSummary():String
setSummary(String):void
<<Java Class>>
UserProfileServiceImpl
com.myy803.socialbookstore.services
searchFactory: SearchFactory
UserProfileServiceImpl()
retrieveProfile(String):UserProfileFormData
save(UserProfileFormData):void
retrieveBookOffers(String):List<BookFormData>
addBookOffer(String,BookFormData):void
searchBooks(SearchFormData):List<BookFormData>
requestBook(int,String):void
retrieveBookRequests(String):List<BookFormData>
retrieveRequestingUsers(int):List<UserProfileFormData>
deleteBookOffer(String,int):void
deleteBookRequest(String,int):void
acceptBookRequest(String,int):void
findBookById(int):BookFormData
convertToUserProfile(UserProfileFormData):UserProfile
convertToUserProfileFormData(UserProfile):UserProfileFormData
convertToBookFormData(Book):BookFormData
-userProfileMapper
0..1
<<Java Interface>>
UserProfileMapper
com.myy803.socialbookstore.mappers
findByUsername(String):UserProfile
-bookAuthorMapper
0..1
<<Java Interface>>
BookAuthorMapper
com.myy803.socialbookstore.mappers
findByName(String):BookAuthor
-bookMapper
0..1
<<Java Interface>>
BookMapper
com.myy803.socialbookstore.mappers
findByTitle(String):List<Book>
findByTitleContaining(String):List<Book>
-bookCategoryMapper
0..1
<<Java Interface>>
BookCategoryMapper
com.myy803.socialbookstore.mappers
findByName(String):BookCategory


<<Java Class>>
AuthController
com.myy803.socialbookstore.controllers
AuthController()
login():String
register(Model):String
registerUser(User,Model):String
~userService
0..1
<<Java Interface>>
UserService
com.myy803.socialbookstore.services
saveUser(User):void
isUserPresent(User):boolean
findById(int):User
<<Java Enumeration>>
Role
com.myy803.socialbookstore.domainmodel
USER: Role
ADMIN: Role
value: String
Role(String)
getValue():String
<<Java Class>>
UserServiceImpl
com.myy803.socialbookstore.services
bCryptPasswordEncoder: BCryptPasswordEncoder
UserServiceImpl()
saveUser(User):void
isUserPresent(User):boolean
loadUserByUsername(String):UserDetails
findById(int):User
-role
0..1
<<Java Class>>
User
com.myy803.socialbookstore.domainmodel
userid: int
username: String
password: String
User()
getAuthorities():Collection<? extends GrantedAuthority>
getUserid():int
setUserid(int):void
getPassword():String
setUsername(String):void
getUsername():String
setPassword(String):void
getRole():Role
setRole(Role):void
isAccountNonExpired():boolean
isAccountNonLocked():boolean
isCredentialsNonExpired():boolean
isEnabled():boolean
-userMapper
0..1
<<Java Interface>>
UserMapper
com.myy803.socialbookstore.mappers
findByUsername(String):User

Class Name: AuthController	
Responsibilities:	**Collaborations:**
▪ Contains the mapping for the login and register use cases	▪ UserService ▪ User ▪ Role

Class Name: UserController	
Responsibilities:	**Collaborations:**
▪ Contains the mapping for the rest use cases	▪ UserProfileService ▪ BookFormData ▪ UserProfileFormData ▪ SearchFormData

Class Name: Book	
Responsibilities:	**Collaborations:**
▪ Contains the fields for the book class	▪ BookAuthor ▪ BookCategory ▪ UserProfile

Class Name: BookAuthor	
Responsibilities:	**Collaborations:**
▪ Contains the fields for the author class	▪ Book

Class Name: BookCategory	
Responsibilities:	**Collaborations:**
▪ Contains the fields for the category class	▪ Book

Class Name: Role	
Responsibilities: ▪ Enum that specifies the role of the user	**Collaborations:** ▪ -

Class Name: User	
Responsibilities: ▪ Contains the proper fields for the user register	**Collaborations:** ▪ Role

Class Name: UserProfile	
Responsibilities: ▪ Contains the fields for the user profile	**Collaborations:** ▪ Book ▪ BookAuthor ▪ BookCategory

Class Name: BookFormData	
Responsibilities: ▪ Contains the fields for the book that is used in the controller section in order not to have direct access to domain models	**Collaborations:** ▪ -

Class Name: UserProfileFormData	
Responsibilities: ▪ Contains the fields for the user profile that is used in the controller section in order not to have direct access to domain models	**Collaborations:** ▪ -

Class Name: SearchFormData	
Responsibilities: ▪ Contains the fields for the search form that is used in the controller section in order not to have direct access to domain models	**Collaborations:** ▪ -

Class Name: BookAuthorMapper	
Responsibilities: ▪ Extends the JpaRepository and its use is to access the database for BookAuthor objects	**Collaborations:** ▪ BookAuthor

Class Name: BookCategoryMapper	
Responsibilities: ▪ Extends the JpaRepository and its use is to access the database for BookCategory objects	**Collaborations:** ▪ BookCategory

Class Name: BookMapper	
Responsibilities: ▪ Extends the JpaRepository and its use is to access the database for Book objects	**Collaborations:** ▪ Book

Class Name: UserMapper	
Responsibilities: ▪ Extends the JpaRepository and its use is to access the database for User objects	**Collaborations:** ▪ User

Class Name: UserProfileMapper	
Responsibilities: ▪ Extends the JpaRepository and its use is to access the database for UserProfile objects	**Collaborations:** ▪ UserProfile

Class Name: UserProfileService	
Responsibilities: ▪ This class is an interface that contains all the actions of the app except the login/register.	**Collaborations:** ▪ UserProfileFormData ▪ BookFormData ▪ SearchFormData

Class Name: UserProfileServiceImpl	
Responsibilities: ▪ Implements the UserProfileService interface ▪ Implements all the actions of the interface	**Collaborations:** ▪ Book ▪ BookAuthor ▪ BookCategory ▪ UserProfile ▪ BookFormData ▪ UserProfileFormData ▪ SearchFormData ▪ BookAuthorMapper ▪ BookCategoryMapper ▪ BookMapper ▪ UserProfileMapper ▪ SearchFactory ▪ SearchStrategy

Class Name: UserService	
Responsibilities: ▪ This class is an interface that contains the action for login and register.	**Collaborations:** ▪ User

Class Name: UserServiceImpl	
Responsibilities: ▪ Implements the UserService interface. ▪ Implements all the actions of the interface.	**Collaborations:** ▪ User ▪ UserMapper

Class Name: SearchFactory	
Responsibilities: ▪ Takes a string input and returns the proper object that implements SearchStrategy.	**Collaborations:** ▪ ExactSearchStrategy ▪ ApproximateSearchStrategy

Class Name: SearchStrategy	
Responsibilities: ▪ This class is an interface for search action.	**Collaborations:** ▪ BookFormData ▪ SearchFormData

Class Name: TemplateSearchStrategy	
Responsibilities: ▪ This is an abstract class that implements SearchStrategy interface. ▪ Contains the implementation for search action. ▪ Contains two abstract classes.	**Collaborations:** ▪ Book ▪ BookFormData ▪ SearchFormData ▪ BookMapper

Class Name: ExactSearchStrategy	
Responsibilities:	**Collaborations:**
▪ Implements the abstract classes from TemplateSearchStrategy	▪ Book ▪ BookAuthor ▪ SearchFormData ▪ BookMapper

Class Name: ApproximateSearchStrategy	
Responsibilities:	**Collaborations:**
▪ Implements the abstract classes from TemplateSearchStrategy	▪ Book ▪ BookAuthor ▪ SearchFormData ▪ BookMapper