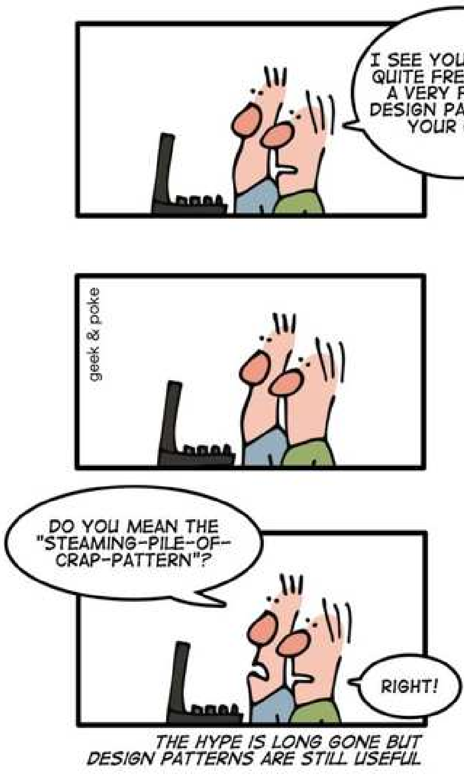
Online Social Book Store Application

Guidelines and Hints for the development of the project


I SEE YOU'VE USED QUITE FREQUENTLY A VERY FAMOUS DESIGN PATTERN IN YOUR CODE
geek & poke
DO YOU MEAN THE "STEAMING-PILE-OF-CRAP-PATTERN"?
RIGHT!
THE HYPE IS LONG GONE BUT DESIGN PATTERNS ARE STILL USEFUL

1 Introduction

This document provides some basic but important guidelines for the development of the project. We begin with general development tips that should be followed. Then, we provide design directions concerning the project.

2 General Guidelines - DOs and DON'Ts

- Classes
  - ✓ Make **classes small** and cohesive - A single well-defined responsibility for a class
  - ✓ Don't break **encapsulation** by making the data representation public
  - ✓ **Class names** are important – use descriptive names for the concepts represented by the classes
  - ✓ Use **Noun & Noun phrases** for class names
  - ✓ **See here for more - http://www.cs.uoi.gr/~zarras/soft-devII.htm**
- Methods
  - ✓ Make **methods small** – A method must **do one thing**
  - ✓ **Method names** are important – use descriptive names for the concepts represented by the methods
  - ✓ Use **Verb & Verb phrases** for method names
  - ✓ **See here for more - http://www.cs.uoi.gr/~zarras/soft-devII.htm**
- Fields
  - ✓ Make **fields private** – A method must do one thing
  - ✓ **Field names** are important – use descriptive names for the concepts represented by the fields
  - ✓ Use **Noun & Noun phrases** for field names
- Follow the standard Java Coding Style **http://www.cs.uoi.gr/~zarras/soft-devII-notes/java-programming-style.pdf**

3 Application Design and Related Design Patterns

3.1 Architecture

IMPORTANT NOTICE: the design that is given here is a **draft/incomplete** version of a social bookstore application prototype that relies on the Spring Boot framework. You can use it as a starting point and follow it to a certain degree. **Feel free to adapt** what is described in this document to your needs and vision.

To facilitate the maintenance and enable future extensions of the application we assume an architecture that relies on Martin Fowler's catalog of **Enterprise Application Architecture (EAA) patterns** (see https://martinfowler.com/eaaCatalog/). Maintainability and extensibility are further promoted by the fact that the **Spring Boot framework** heavily relies on the **Model View Controller (MVC) pattern (see the lecture slides on Software Design)** for the development of Web applications. MVC allows the clear separation of three different concerns: the view of the application that is responsible for the user interaction (UI) with the application, the domain model that represents the data handled by the application and the business logic, and the controller that takes user input, manipulates the domain model data and updates the view.

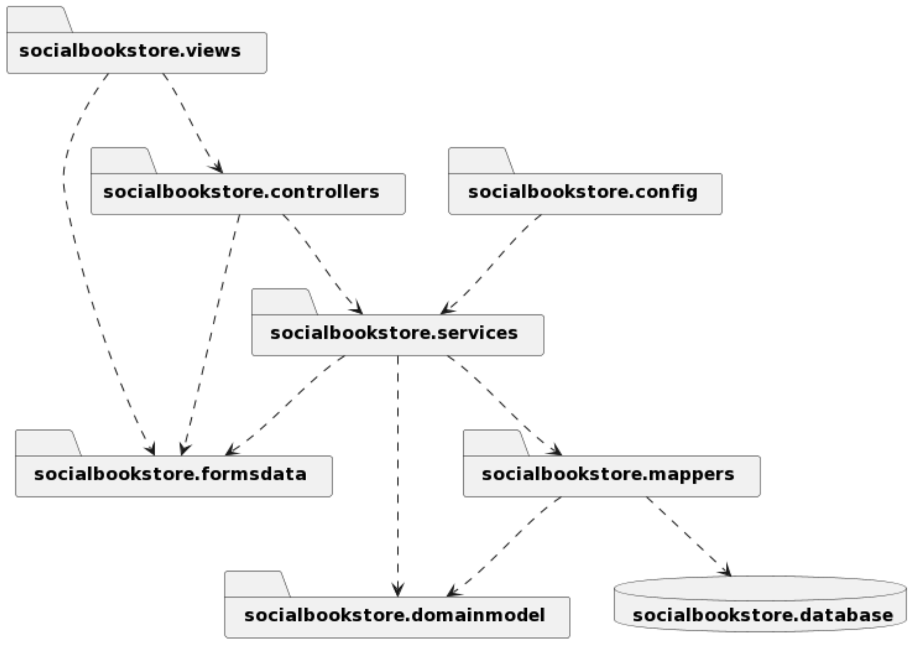


FIGURE 1 APPLICATION ARCHITECTURE.

Figure 1, illustrates a draft architecture for the application. The following list describes briefly the basic components of the architecture (Figure 1):

- The **views package** realizes the user interaction (UI) with the application in collaboration with the Controller layer. Typically, this layer consists of a set of static and dynamic Web pages. The dynamic Web pages are also known as template views (see **Template View pattern** from Martin Fowler's catalog of EAA patterns). A dynamic HTML page renders data from domain model objects into HTML based on embedded markers. The application controllers is are responsible for passing the appropriate domain model objects to the view layer.

- The **controllers package** consists of controller classes which take user input from the views of the application, perform certain **service operations** (see next) to manipulate domain model objects, and update the views of the application.
- The **formsdata package** comprises classes that are used for transferring data to/from the views from/to the back end of the application. Having such classes, isolates the views from the **model** (see next) and the business logic of the application. Moreover, it eases the manipulation of the data from the template engine that is responsible for the views' generation.
- The **services package** defines a set of services provided by the application to the users and coordinates the application's response in each service operation. Essentially, here we apply the **Service Layer pattern** from Martin Fowler's catalog of EAA patterns.
- The **mappers package** consists of classes that perform basic database operations which map the application data, stored in the table rows of the database to corresponding in memory objects of the domain model classes. Basically, here we apply the **Data Mapper pattern** from **Martin Fowler's catalog of EAA patterns**.
- The backbone of the architecture is the **domain model package**. The domain model consists of the basic classes that define the representation of the data handled by the application and the domain logic that is needed for the data manipulation. All the different layers of the application rely on the data model.

3.2 Domain Model

The **domain model** that we assume in this draft prototype is given in Figure 2. **User** and **Role** are specific classes we must implement for the realization of the user registration and login actions in **Spring Boot**. User should implement the Spring **UserDetails** interface for this reason [1].

[1] https://github.com/zarras/myy803_springboot_web_app_tutorials/tree/master/sb_tutorial_7_signup_signin

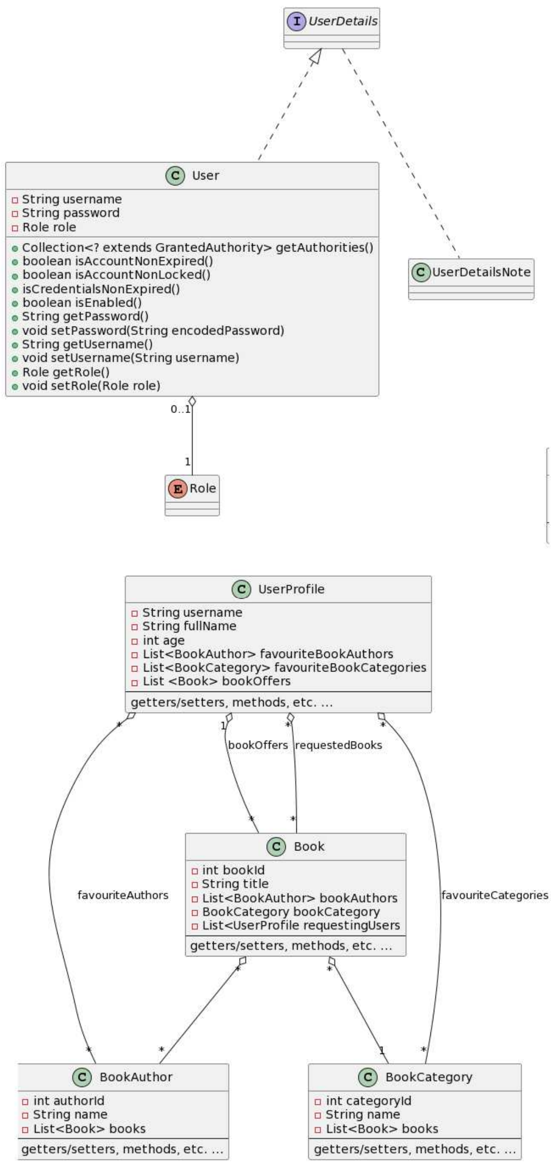

UserDetails
User
String username
String password
Role role
Collection<? extends GrantedAuthority> getAuthorities()
boolean isAccountNonExpired()
boolean isAccountNonLocked()
isCredentialsNonExpired()
boolean isEnabled()
String getPassword()
void setPassword(String encodedPassword)
String getUsername()
void setUsername(String username)
Role getRole()
void setRole(Role role)
UserDetailsNote
0..1
1
Role
UserProfile
String username
String fullName
int age
List<BookAuthor> favouriteBookAuthors
List<BookCategory> favouriteBookCategories
List <Book> bookOffers
getters/setters, methods, etc. ...
bookOffers
requestedBooks
favouriteAuthors
favouriteCategories
Book
int bookId
String title
List<BookAuthor> bookAuthors
BookCategory bookCategory
List<UserProfile requestingUsers
getters/setters, methods, etc. ...
BookAuthor
int authorId
String name
List<Book> books
getters/setters, methods, etc. ...
BookCategory
int categoryId
String name
List<Book> books
getters/setters, methods, etc. ...

FIGURE 2 DOMAIN MODEL.

UserProfile is responsible for the management of the users' profiles. Specifically, the class fields include the username the full name of the user the age of the user, a list of books offered by the user, a list of books requested by the user, a list of favorite book authors and a list of favorite book categories. The relation between UserProfile and BookAuthor is many to many, the relation between UserProfile and BookCategory is also many to many. The requestedBooks relation is also many to many, while the bookOffers relation is one to many. The relation between Book and BookAuthor is many to many, while the relation between Book and BookCategory is many to one. The database schema of the application defines corresponding tables for the classes of the application with foreign keys that correspond to the class associations. In Spring Boot the SQL schema can be automatically generated from the domain model with the appropriate JPA annotations [2].

3.3 Data Mappers

The mappers package consists of several interface definitions derived from the general JpaRepository interface that is provided by Spring Boot. Specifically, the contents of the package are given in Figure 2.

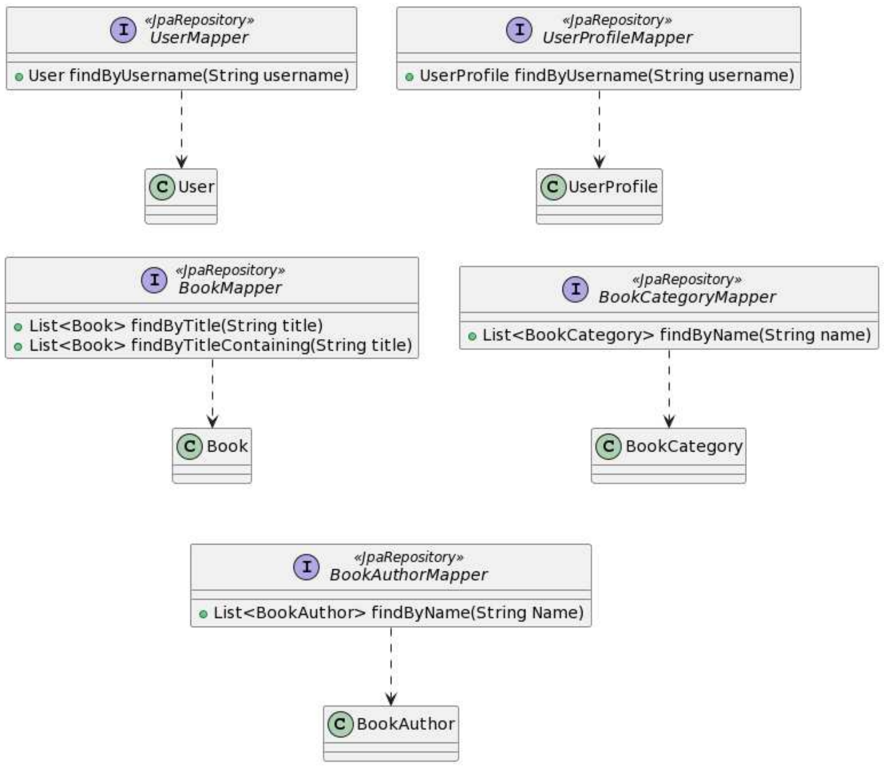

«JpaRepository» UserMapper
User findByUsername(String username)
User
«JpaRepository» UserProfileMapper
UserProfile findByUsername(String username)
UserProfile
«JpaRepository» BookMapper
List<Book> findByTitle(String title)
List<Book> findByTitleContaining(String title)
Book
«JpaRepository» BookCategoryMapper
List<BookCategory> findByName(String name)
BookCategory
«JpaRepository» BookAuthorMapper
List<BookAuthor> findByName(String Name)
BookAuthor

FIGURE 3 DATA MAPPERS.

3.4 Services

The first service that is provided by the online social bookstore application is responsible for the user registration and login (Figure 4). The service provides the UserService interface that is implemented by

2 https://github.com/zarras/myy803_springboot_web_app_tutorials, https://github.com/zarras/myy803_springboot_jpa_tutorials, https://www.baeldung.com/jpa-many-to-many

the UserServiceImpl class. The service further implements the UserDetailsService interface, as required by the Spring Security framework.

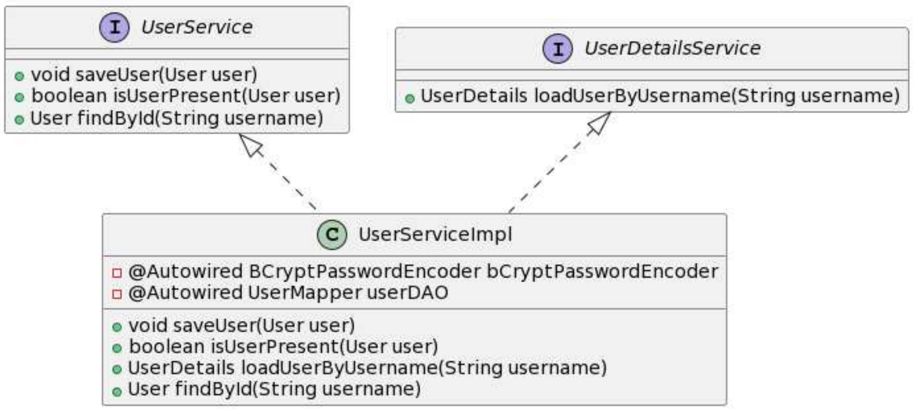

UserService
void saveUser(User user)
boolean isUserPresent(User user)
User findById(String username)
UserDetailsService
UserDetails loadUserByUsername(String username)
UserServiceImpl
@Autowired BCryptPasswordEncoder bCryptPasswordEncoder
@Autowired UserMapper userDAO
void saveUser(User user)
boolean isUserPresent(User user)
UserDetails loadUserByUsername(String username)
User findById(String username)

FIGURE 4 USER SERVICE.

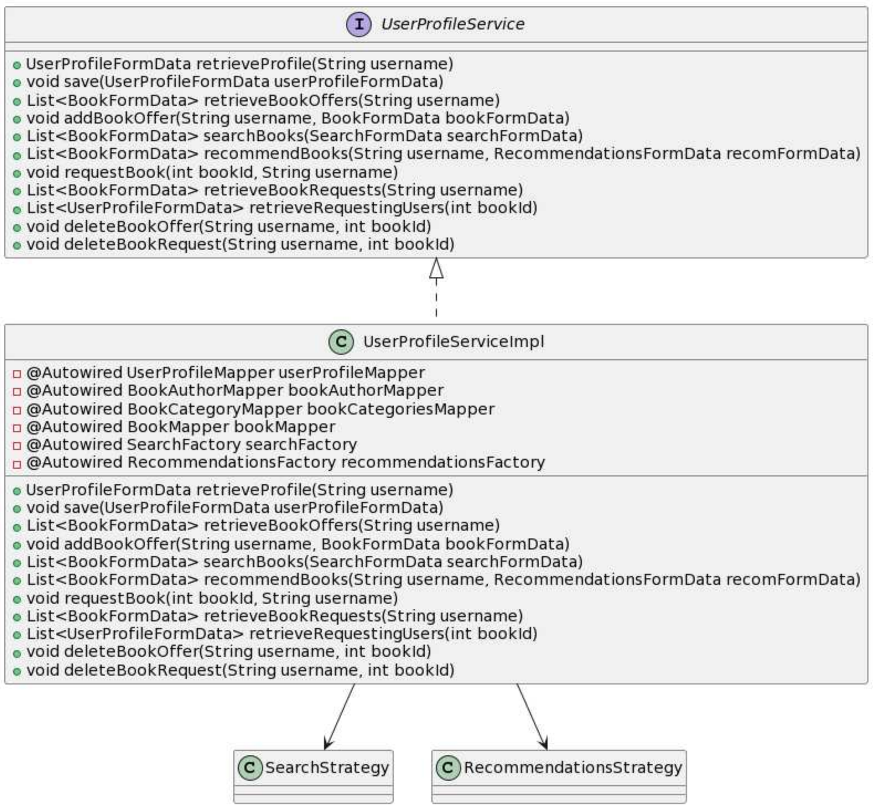

UserProfileService
UserProfileFormData retrieveProfile(String username)
void save(UserProfileFormData userProfileFormData)
List<BookFormData> retrieveBookOffers(String username)
void addBookOffer(String username, BookFormData bookFormData)
List<BookFormData> searchBooks(SearchFormData searchFormData)
List<BookFormData> recommendBooks(String username, RecommendationsFormData recomFormData)
void requestBook(int bookId, String username)
List<BookFormData> retrieveBookRequests(String username)
List<UserProfileFormData> retrieveRequestingUsers(int bookId)
void deleteBookOffer(String username, int bookId)
void deleteBookRequest(String username, int bookId)
UserProfileServiceImpl
@Autowired UserProfileMapper userProfileMapper
@Autowired BookAuthorMapper bookAuthorMapper
@Autowired BookCategoryMapper bookCategoriesMapper
@Autowired BookMapper bookMapper
@Autowired SearchFactory searchFactory
@Autowired RecommendationsFactory recommendationsFactory
UserProfileFormData retrieveProfile(String username)
void save(UserProfileFormData userProfileFormData)
List<BookFormData> retrieveBookOffers(String username)
void addBookOffer(String username, BookFormData bookFormData)
List<BookFormData> searchBooks(SearchFormData searchFormData)
List<BookFormData> recommendBooks(String username, RecommendationsFormData recomFormData)
void requestBook(int bookId, String username)
List<BookFormData> retrieveBookRequests(String username)
List<UserProfileFormData> retrieveRequestingUsers(int bookId)
void deleteBookOffer(String username, int bookId)
void deleteBookRequest(String username, int bookId)
SearchStrategy
RecommendationsStrategy

FIGURE 5 USER PROFILE SERVICE.

The second service of the online social bookstore application is responsible for the rest of the user stories, i.e., the user profile management, the book offers, the book requests, the search and the recommendation of book offers (Figure 6). The key elements of this service are the UserProfileService interface and the respective implementation UserProfileServiceImpl class. Alternatively, this service can be split into multiple smaller more cohesive services that assume the different responsibilities.

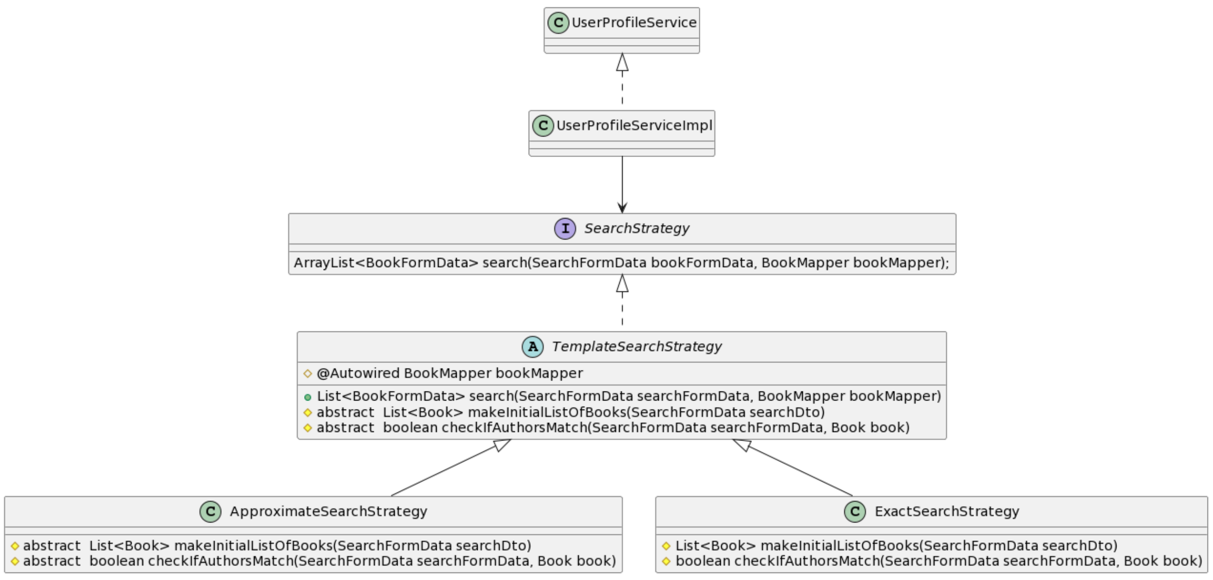

UserProfileService
UserProfileServiceImpl
SearchStrategy
ArrayList<BookFormData> search(SearchFormData bookFormData, BookMapper bookMapper);
TemplateSearchStrategy
@Autowired BookMapper bookMapper
List<BookFormData> search(SearchFormData searchFormData, BookMapper bookMapper)
abstract List<Book> makeInitialListOfBooks(SearchFormData searchDto)
abstract boolean checkIfAuthorsMatch(SearchFormData searchFormData, Book book)
ApproximateSearchStrategy
abstract List<Book> makeInitialListOfBooks(SearchFormData searchDto)
abstract boolean checkIfAuthorsMatch(SearchFormData searchFormData, Book book)
ExactSearchStrategy
List<Book> makeInitialListOfBooks(SearchFormData searchDto)
boolean checkIfAuthorsMatch(SearchFormData searchFormData, Book book)

FIGURE 6 BOOK OFFERS SEARCH STRATEGIES.

For the realization of the book offers search and recommendation we rely on the GoF **strategy pattern** and the GoF **template method** pattern (Figure 6). The basic reason for these design choices is the **maintainability requirements** given in the application requirements document for the **easy extension of the application with new strategies** in the future.

Specifically, the idea is to define a common SearchStrategy interface for the different search strategies (see GoF strategy pattern). Moreover, we assume an abstract class TemplateSearchStrategy that implements the common interface and realizes the basic search algorithm. The steps of the search algorithm that differ from strategy to strategy are defined as abstract methods, implemented by respective subclasses, ApproximateSearchStrategy, ExactSearchStrategy (see GoF template method pattern). The design of the book offer recommendations strategies can rely on the same ideas and is not further discussed here.

3.5 MVC- Controllers, Views, and Form Data

The controllers package comprises two different controllers. The AuthController is responsible for the user registration and login. The UserProfileController is responsible for the rest of the user stories. Alternatively, the UserProfileController can be split into a number of smaller more cohesive controllers that assume the different responsibilities. The controllers rely on the respective services of the services package for the manipulation of domain objects and the update of the views. The views are based on the Thymeleaf template engine. Data transfer between the views, the controllers and the services is done via objects that belong to the classes of the formsdata package. In particular, the UserProfileFromData, the BookFormData, the SearchFormData and the RecommendationsFormData are used for transferring data that concern user profiles, books, search and recommendation requests, respectively.

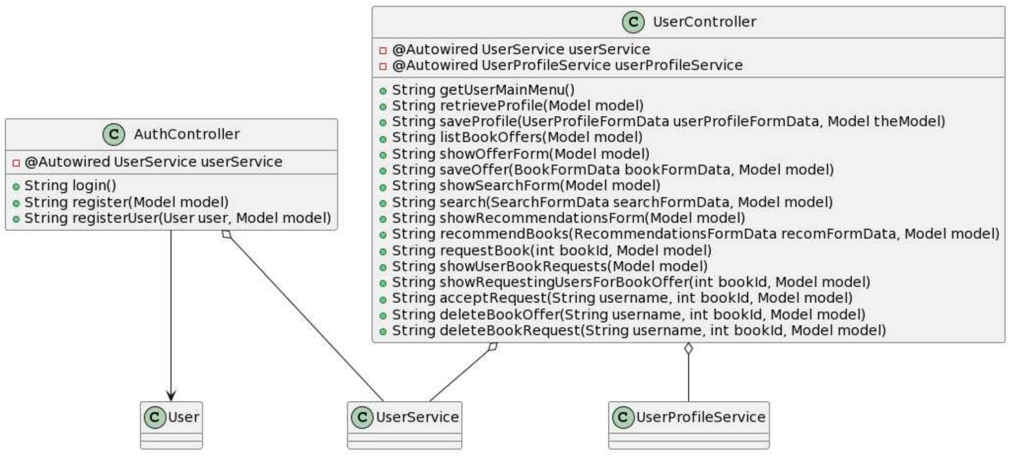

UserController
@Autowired UserService userService
@Autowired UserProfileService userProfileService
String getUserMainMenu()
String retrieveProfile(Model model)
String saveProfile(UserProfileFormData userProfileFormData, Model theModel)
String listBookOffers(Model model)
String showOfferForm(Model model)
String saveOffer(BookFormData bookFormData, Model model)
String showSearchForm(Model model)
String search(SearchFormData searchFormData, Model model)
String showRecommendationsForm(Model model)
String recommendBooks(RecommendationsFormData recomFormData, Model model)
String requestBook(int bookId, Model model)
String showUserBookRequests(Model model)
String showRequestingUsersForBookOffer(int bookId, Model model)
String acceptRequest(String username, int bookId, Model model)
String deleteBookOffer(String username, int bookId, Model model)
String deleteBookRequest(String username, int bookId, Model model)
AuthController
@Autowired UserService userService
String login()
String register(Model model)
String registerUser(User user, Model model)
User
UserService
UserProfileService

FIGURE 7 CONTROLLERS

4 Tests

Concerning the tests of the application logic we need to develop tests for the different packages of the application (mappers, services, controllers, model). For the development of the tests we can rely on the SpringBoot testing and mocking facilities.