Αναπτυξη Λογισμικου 2

2024-2025

Κωνσταντίνος-Διονύσιος Λαμανιάκου
ΑΜ 5110

Ελένη Αγγελοπούλου
ΑΜ 4968

ONLINE SOCIAL BOOKSTORE APPLICATION REENGINEERING THE LEGACY CODE

TABLE OF CONTENTS

Introduction

The objective of this project phase is to reengineer an existing online social bookstore application developed with Spring Boot in Java. The primary goal is to enhance the application's structure and maintainability by redistributing responsibilities, addressing duplicate code, and separating business logic from user interaction. Key tasks include:

1. **Code Review**: Analyzing the legacy code to identify areas for improvement in readability, abstraction, and functionality.

2. **Initial Understanding**: Reviewing documentation and the application's database schema to understand its core design and functionality.

3. **Refactoring** :Implementing improvements by refactoring classes, notably restructuring the UserController to separate user profile, book management, and search functionalities.

4. **Testing**: Introducing unit and integration tests using JUnit, Mockito, and Spring Boot Testing frameworks to ensure the integrity of the refactored code.

5. **Enhancements**: Extending functionality by adding new search and recommendation strategies to better support user needs.

This phase will conclude with the submission of a detailed report on findings, class diagrams, and test implementations to document the reengineering efforts.

First Contact

Legacy System - Source code Notes

1. Coding Conventions

- **Naming**: Method and variable names are quite descriptive (e.g., showOfferForm, saveProfile, deleteBookRequest ,saveUser, isUserPresent, loadUserByUsername), which makes the code easier to understand.
- Constants are not used for repeated messages (e.g., `"USER_NOT_FOUND"`). Extracting such strings to constants would be beneficial to prevent errors and make the code easier to update.

2. Abstractions and Interfaces

- **Separation of Responsibilities**: The UserController class has many methods, increasing complexity and making it hard to maintain. Splitting functionality across different controllers for user profiles, book offers, and requests would help improve organization.
- **Reusable Strategies**: The use of SearchFactory and RecommendationsFactory is a good design choice that allows for extending functionality based on the user's preferences, effectively separating search and recommendation logic.
- The class UserServiceImpl implements two interfaces: UserService and UserDetailsService. This shows good abstraction since it separates general user service functionality from the Spring Security-specific interface for user details.

3. Code Smells

- **Duplicate Code**: There is repeated code for retrieving the username from SecurityContextHolder, such as in methods like retrieveProfile, saveOffer, listBookOffers etc. A helper method to retrieve the username would improve readability and reduce repetition.
- **String Format Issue**: The String.format(`"USER_NOT_FOUND"`, username) seems to be incorrectly used, as no placeholder (%s) is provided for username. This could lead to a UsernameNotFoundException that doesn't convey the username involved in the error.

4. Important Comments

- **TODO Comments for Future Features**: There are TODO comments in some parts of the code (e.g., accept_request, deleteBookOffer). This is useful as it highlights unimplemented functionality for other developers.
- **Lack of Comments**: The code lacks comments explaining the purpose of each method and class. Adding brief comments for methods like `saveUser` (explaining that it encrypts and saves the user's password) would improve readability, especially for developers who are unfamiliar with the code.
- **Error Handling**: Adding comments to clarify why certain exceptions (e.g., `UsernameNotFoundException`) are thrown can make the code easier to maintain and debug.
- **Lack of Explanatory Comments**: Some methods could benefit from additional comments. Although method names are descriptive, a few explanatory comments on complex processes (e.g., recommend) would enhance understanding.

5. Availability of Tests

- **Lack of Tests**: The code doesn't have any tests. Having tests for core functionalities like retrieveProfile, saveOffer, and search is essential to ensure that the endpoints work correctly.

Reengineering Feasibility

Reengineering the code would be beneficial

- Breaking down the controller into smaller controllers
- Reducing code duplication
- Adding unit tests
- Moving constant strings like `"USER_NOT_FOUND"` to a centralized place.
- Adding error handling improvements and clarifying comments.

would improve maintainability and structure.

Legacy System - Documentation Notes

1. Requirements Definition (RequirementsDefinition-2024-v05-03-24.pdf)

Functional Requirements/User Stories

- **Why Useful:**
  They outline what the system must do, including features like user registration, book offering, searching, and recommendations. These are essential for understanding the core functionalities to maintain or improve.
- **Update Status:**
  Appears mostly up-to-date but may lack coverage of edge cases or newer enhancements. Ensure implemented features fully match the documented user stories.

Non-Functional Requirements

- **Why Useful:**
  Emphasizes maintainability and usability. Concepts like low coupling, high cohesion, and extensible strategies (e.g., search and recommendation strategies) guide architectural improvements.
- **Update Status:**
  Current, but implementation adherence to these principles should be verified.
  Technical Constraints/Recommendations.

Technical Constraints/Recommendations

- **Why Useful:**
  Provides a foundation for understanding the tools and frameworks used (e.g., Spring Boot, MySQL, JUnit, Mockito). Ensures reengineering aligns with the existing tech stack.
- **Update Status:**
  Likely up-to-date, as technologies like Spring Boot and MySQL remain relevant.

2. Guidelines and Hints (GuidelinesAndHints-18-09-24.pdf)

Application Design

- **Why Useful:**
  Explains the use of MVC architecture, domain modeling, and design patterns like Strategy and Data Mapper. This is critical for maintaining system consistency while reengineering.
- **Update Status:**
  Up-to-date conceptually, though specific implementation details (e.g., additional domain relationships or template engine updates) might have changed.

Implementation Guidance

- **Why Useful:**
  Describes the responsibilities of key components (e.g., UserMapper, UserServiceImpl) and how they interact. Helps identify dependencies for reengineering.
- **Update Status:**
  Likely current for major components but should be cross-checked with the actual codebase.

Installation and Execution

- **Why Useful:**
  Details steps for setting up the system, including database configuration and Maven-based builds. Useful for environment replication during reengineering.
- **Update Status:**
  Appears up-to-date but dependent on project-specific updates, such as new dependencies.

TODO Section

- **Why Useful:**
  Points out missing features (e.g., deletion of book offers, test base) and areas needing improvement. These could be prioritized in reengineering efforts.
- **Update Status:**
  Reflects the state of the preliminary version. Some tasks may have been completed since its creation.

Legacy System - Use Cases

1:User Registration

Use case ID	US1
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to database.
Main flow of events	1. The use case starts when an unauthenticated user navigates to the registration page. 2. The user enters the required registration information and clicks the "Register" button. 3. The application validates the entered information for completeness and correctness. 4. The application attempts to create a new user account with the provided credentials in the database. 5. The application redirects to the home page.
Alternative flow 1	1. The alternative flow begins after step 4. 2. The application determines that the username or email is already in use. 3. The application displays a "Username or Email is already in use" message.
Post conditions	The user is registered and can proceed to log in with the newly created credentials.

2:User Logout

Use case ID	US2
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to database.
Main flow of events	1. The use case starts when an unauthenticated user attempts to access any of the applications' pages that require authentication, including the login page itself. 1.1 If the user requests a page other than the login page. 1.2 The application redirects to the login page. 2. The user enters his/her username and password and clicks the "log in" button. 3. The application attempts to authenticate the users credentials (username and password with the respective credentials in the database. 4. The application redirects to the home page.
Alternative flow 1	1. The alternative flow begins after step 3. 2. The application determines the given user credentiais as invalid. 3. The application displays a "Username or Password is invalid." message.
Post conditions	The user is authenticated and can access any of the applications' pages.

3:User Logout

Use case ID	US3
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application.
Main flow of events	1. The use case starts when an authenticated user clicks the "logout" button or link in the application. 2. The application invalidates the user's session. 3. The application redirects the user to the login page.
Post conditions	The user's session is invalidated, and the user is logged out of the application. The user is redirected to the login page and can no longer access authenticated pages without logging in again.

4:User Profile submission

Use case ID	US4
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application.
Main flow of events	1. The use case starts when an authenticated user navigates to the "Profile" page. 2. The user enters their full name, address, age, phone number, preferred book categories, and favorite authors. 3. The user clicks the "Submit" button. 4. The application validates the entered information. 5. The application saves the profile information in the database.

	6. The application redirects the user to the profile view page.
Post conditions	If the profile creation is successful, the user's profile information is stored in the database, and the user can view and edit their profile.

5:Add Book Offer

Use case ID	US5
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application.
Main flow of events	1. The use case starts when an authenticated user navigates to the "My Books/AddBook" page. 2. The user enters the book title, author(s), category. 3. The user clicks the "Submit" button. 4. The application saves the book offer information in the database. 5. The application redirects the user to their personal list of book offers.
Post conditions	If the book offer is added successfully, the book offer information is stored in the database, and the user can view it in their personal list of book offers.

6:Browse Requests for Book Offer

Use case ID	US6
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. A User has at least one book offer listed.
Main flow of events	1. The use case starts when an authenticated user navigates to their list of book offers"My Books". 2. The user select "Requests" a specific book offer from their list. 3. The application retrieves a list of requests from other users interested in the selected book offer from the database including actions such as take the book and Contact information . 4. The user reviews the list of requests and can make a decision on whom to give the book.
Post conditions	The user is able to see a list of requests from other users interested in their book offer and can make an informed decision on whom to give the book.

7:Notify Users about Book Offer

Use case ID	US7
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. User has at least one book offer listed. 6. There are requests from other users for the book offer.
Main flow of events	1. The use case starts when an authenticated user navigates to their list of book offers. 2. The user selects a specific book offer from their list.
	3. The application retrieves a list of requests from other users interested in the selected book offer from the database. 4. The user selects a specific request from the list. 5. The user clicks the "Take the book" button for a specific user. 6. The application sends a notification to the selected user informing them that they can take the book. 7. The application sends notifications to the other users who requested the book, informing them that the book has been taken by another user.
Post conditions	If the notifications are sent successfully, the selected user is informed that they can take the book, and the rest of the users are informed that the book has been taken by another user.

8:Access Contact Information of Requester

Use case ID	US8
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. User has at least one book offer listed. 6. There are requests from other users for the book offer.
Main flow of events	1. The use case starts when an authenticated user navigates to their list of book offers. 2. The user selects a specific book requests action from their list. 3. The application retrieves a list of requests from other users interested in the selected book offer from the database. 4. The user selects a specific contact information from the list. 5. The application displays the contact information of the selected user (e.g., name, email, phone number). 6. The user uses the contact information to arrange the delivery of the book.
Post conditions	The user successfully accesses the contact information of the selected requester and can contact them to arrange the delivery of the book.

9:Remove Book Offer

Use case ID	US9
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. User has at least one book offer listed.
Main flow of events	1. The use case starts when an authenticated user navigates to their list of book offers. 2. The user selects a specific book offer they want to remove. 3. The user clicks the "Delete" button. 4. The application deletes the book offer from the user's personal list of book offers in the database.
Post conditions	If the book offer is removed successfully, it is deleted from both the user's personal list of book offers and the request lists of other users.that is neceseary for accurate book search/recommendation

10:Browse Search and Request Book Offers

Use case ID	US10
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. A User has at least one book offer listed.
Main flow of events	1. The use case starts when an authenticated user navigates to the "Search for Books" page. 2. The user enters search criteria, such as book title and author. 3. The user selects either an exact or an approximate search strategy. 4. The user clicks the "Search" button. 5. The application validates the search criteria. 6. The application performs the search based on the criteria and selected search strategy. 7. The application displays the search results, which include book offers that match the search criteria. 8. The user reviews the search results and selects a specific book offer. 9. The user clicks the "Request" button for the selected book offer. 10. The application displays a " Are you sure you want to request for this book?" message. 11. The application sends a request to the user who offers the book
Post conditions	If the search and request are successful, the user is able to find book offers that match the search criteria and send a request to the user offering the book..

11:Browse and Request Recommended Book Offers

Use case ID	US11
Actors	User
Pre conditions	1. Application successfully initialized and running on respective host. 2. Database initialized and running on respective host. 3. Application successfully connected to the database. 4. User is authenticated and logged into the application. 5. User has a completed profile with favorite category.
Main flow of events	1. The use case starts when an authenticated user navigates to the "Recommended Books Offers" page. 2. The application retrieves a list of recommended book offers based on the users favorite category. 3. The application displays the list of recommended book offers to the user. 4. The user browses the list and selects a specific recommended book offer. 5. The user clicks the "Request Book" button. 6. The application displays a " Are you sure you want to request for this book ?" message. 7. The application sends a request notification to the user who offers the book, informing them that the user is interested in getting the book.
Post conditions	If the request is sent successfully, the user offering the book is notified of the interest, and the interested user is informed that their request has been sent.

12: Remove Book Request

Use case ID	US12
Actors	User
Pre conditions	1. Application is successfully initialized and running on the respective host. 2. Database is initialized and running on the respective host. 3. The application is successfully connected to the database. 4. The user is authenticated and logged into the application. 5. The user has previously made a book request.
Main flow of events	1. The use case starts when an authenticated user navigates to the "My Book Requests" page. 2. The application retrieves and displays a list of the user's active book requests. 3. The user selects a specific book request they want to remove. 4. The user clicks the "Remove Request" button. 5. The application displays a confirmation message: "Are you sure you want to remove this book request?" 6. The user confirms the removal of the book request. 7. The application removes the book request from the system. 8. The application sends a notification to the user who offers the book, informing them that the request has been withdrawn.
Post conditions	If the request is successfully removed, the user who offers the book is notified that the request is no longer active. The requesting user no longer sees the removed request in their "My Book Requests" list.

Legacy System Installation Notes

1. Project Information

Project Name: socialbookstore

Version: 0.0.1-SNAPSHOT

Parent Framework: Spring Boot 3.2.2

Java Version: 17

2. Dependencies and Libraries

Below are the libraries and frameworks used in the project along with their purposes:

Library	Version	Purpose
Spring Boot Starter Data JPA	3.2.2	ORM for database operations using JPA and Hibernate.
Spring Boot Starter Security	3.2.2	Security and authentication framework.
Spring Boot Starter Thymeleaf	3.2.2	Template engine for dynamic HTML rendering.
Spring Boot Starter Web	3.2.2	Web application development, including REST APIs.
Thymeleaf Extras Spring Security 6	3.2.2	Thymeleaf integration with Spring Security.
MySQL Connector/J	8.0.x	JDBC driver for connecting the application to MySQL database.
Lombok	1.18.x	Reduces boilerplate code (e.g., getters, setters) in Java classes.
Spring Boot Starter Test	3.2.2	Testing utilities, including JUnit and Mockito support.
Spring Security Test	3.2.2	Testing utilities specifically for Spring Security.
Spring Boot DevTools	3.2.2	Development utilities for hot reloading and faster testing during dev cycle.

3. Build Configuration

- **Build Tool**: Maven
- **Plugins**:`spring-boot-maven-plugin`: Facilitates building and packaging the Spring Boot application into an executable JAR file.

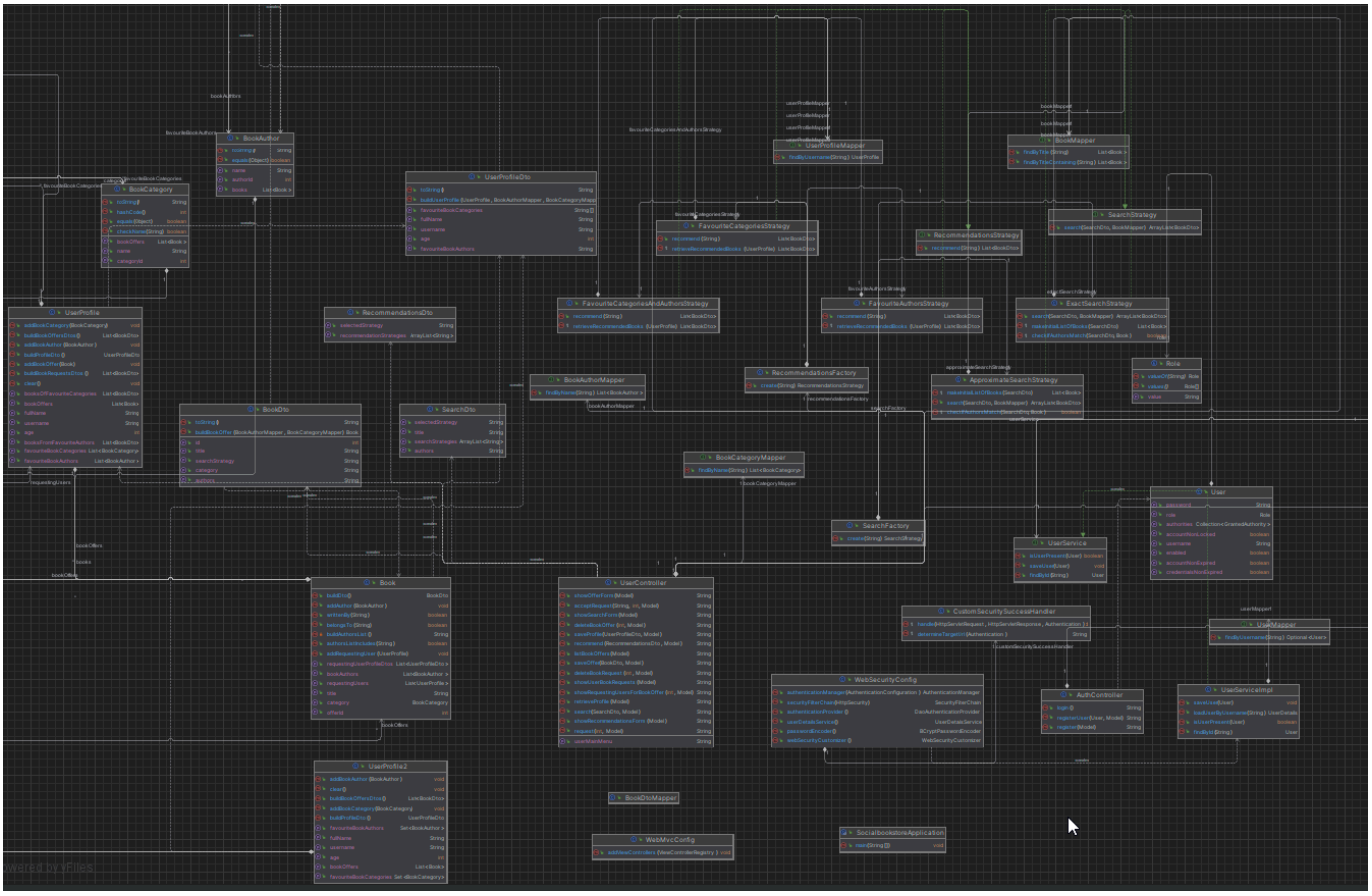
4.Database Configuration

- **Database Type**: MySQL
- **Connection String**: `jdbc:mysql://localhost:3306/socialbookstore`
- **Database Username**: root
- **Database Password**: root

5.Server Configuration

- **Application Port**: `8080` .

Initial Understanding and Detailed Model Capture



Key Classes and Their Responsibilities

1. **User Profiles**
   - **Responsibilities**: Represents individual users, storing personal information like `username`, `age`, and `full_name`.
   - **Relations**:
     - Linked to `Books` as the `profile_username` to track books uploaded by the user.
     - Has many-to-many relationships with `Book Categories` and `Book Authors` to store user interests.
2. **Users**
   - **Responsibilities**: Handles authentication and authorization with fields like `username`, `password`, and `role`.
   - **Relations**: Shares a primary key (`username`) with `User Profiles`, ensuring a one-to-one mapping.

3. **Books**
    - **Responsibilities**: Represents book entries with details like `title`, `category_id`, and `profile_username` for ownership.
    - **Relations**:
        - Linked to `Book Categories` by `category_id`.
        - Linked to `User Profiles` to track the owner.
        - Linked to `Book Authors` through a many-to-many relationship (`Books_Authors`).
4. **Book Authors**
    - **Responsibilities**: Stores information about authors.
    - **Relations**:
        - Linked to `Books` via the join table `Books_Authors`.
        - Linked to `User Profiles` to show user-author associations.
5. **Book Categories**
    - **Responsibilities**: Stores categories or genres of books.
    - **Relations**:
        - Linked to `Books`.
        - Has a many-to-many relationship with `User Profiles`.
6. **Books_Authors** (Join Table)
    - **Responsibilities**: Connects `Books` and `Book Authors` for a many-to-many relationship.
    - **Relations**: References `Books` and `Book Authors`.
7. **Books_Requesting_Users**
    - **Responsibilities**: Tracks which users requested which books.
    - **Relations**: Links `Books` and `User Profiles`.
8. **Profiles_Authors**
    - **Responsibilities**: Stores associations between users and their favorite authors.
    - **Relations**: Connects `User Profiles` and `Book Authors`.
9. **Profiles_Categories**
    - **Responsibilities**: Stores associations between users and their preferred book categories.
    - **Relations**: Connects `User Profiles` and `Book Categories`.

Class Diagram Relations Overview

- **One-to-Many**:
  - `Books` ↔ `Book Categories` (one category for multiple books).
  - `Books` ↔ `User Profiles` (each book has one owner).
- **Many-to-Many**:
  - `Books` ↔ `Book Authors` through `Books_Authors`.
  - `User Profiles` ↔ `Book Categories` through `Profiles_Categories`.
  - `User Profiles` ↔ `Book Authors` through `Profiles_Authors`.

Tests

Controller TESTS:

Authentication (AuthControllerTest):

- Verifies the correct rendering of login and registration pages.

User Management (UserControllerTest):

- Tests user profile-related actions, such as redirecting to the user profile page after saving a profile.

Book Management (BookControllerTest):

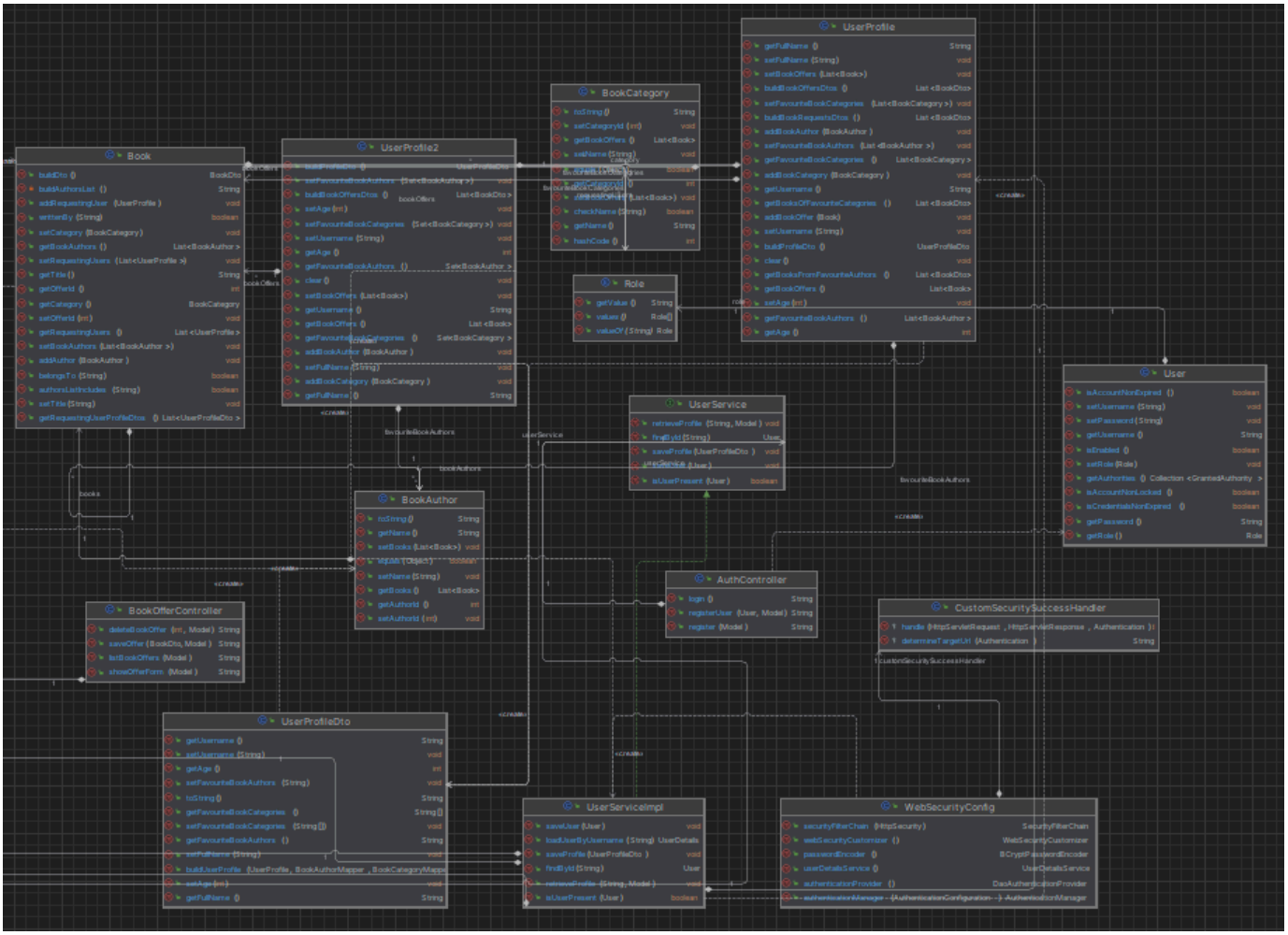
- Ensures that the "Add Book" form is displayed properly and that navigation back to the user dashboard works as expected.

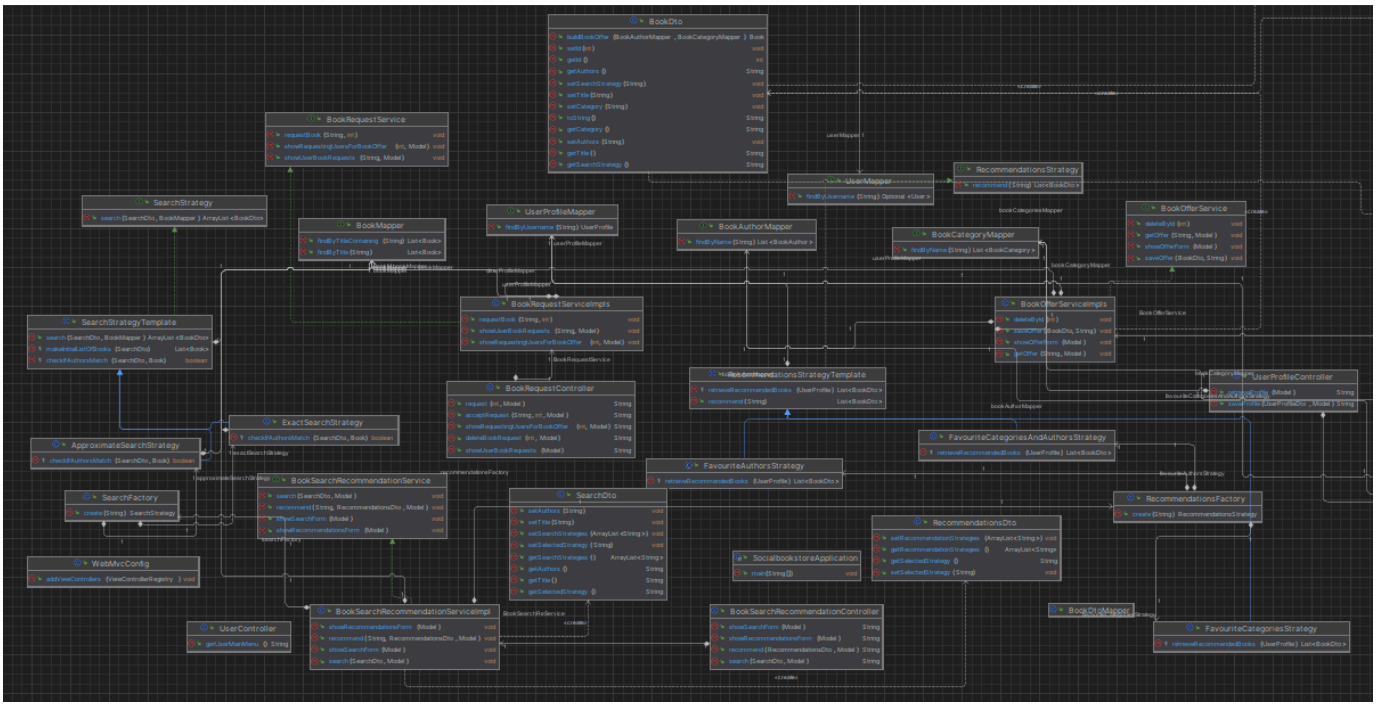
Services TEST:

Explanation of the Tests:

1. **`testSaveUser`**:
   - Ensures that the user password is encoded correctly before being saved.
   - Verifies that the `save` method of `UserMapper` is called.
2. **`testIsUserPresent`**:
   - Checks whether the method correctly identifies when a user exists or does not exist in the database.
3. **`testLoadUserByUsername`**:
   - Confirms that the method returns the correct user when they exist.
   - Ensures that a `UsernameNotFoundException` is thrown if the user does not exist.
4. **`testFindById`**:
   - Similar to `testLoadUserByUsername`, it validates the behavior for finding users by ID.

Reengineered/Extended Design of the Legacy System





Redistribute responsibilities:

Controllers

1. **AuthController**:
   - Manages authentication and authorization-related tasks such as user registration and login.
   - Works closely with `UserService` and security configurations for authentication logic.
2. **BookOfferController**:
   - Handles operations related to book offers, such as creating and retrieving offers.
   - Collaborates with `BookOfferService` to manage business logic for offers.
3. **BookRequestController**:
   - Manages book requests by users, including creating, updating, and tracking requests.
   - Relies on `BookRequestServiceImpl` for handling domain-specific operations.
4. **BookSearchRecommendationController**:
   - Facilitates book search functionality and recommendation logic.
   - Works closely with `BookSearchRecommendationService`, `SearchFactory`, and various `SearchStrategy` implementations.
5. **UserProfileController**:
   - Responsible for managing user profile details.
   - Utilizes `UserService` and `UserProfileMapper` for data access and transformation.

Services

1. **BookOfferService**:
   - Encapsulates business logic for book offers.
   - Uses mappers like `BookMapper` for data translation between domain and DTO.

2. **BookRequestServiceImpl**:
    - Implements functionality for managing book requests.
    - Works with `BookRequestMapper` and `BookRequestDAO` for data persistence.
3. **BookSearchRecommendationService**:
    - Implements the search and recommendation logic.
    - Collaborates with `SearchFactory` and strategies like `ExactSearchStrategy` and `ApproximateSearchStrategy`.
4. **UserService**:
    - Central service for user-related operations such as retrieving profiles and managing roles.
    - Interacts with `UserMapper` and `UserDAO` for data management.

Duplicate code:

Recommendation Strategies:

- These implement recommendation logic based on user preferences like favorite authors or categories.
- Derived from a common base (`RecommendationsStrategy`) and follow a template (`RecommendationsStrategyTemplate`) for extensibility and polymorphism.
- Managed and instantiated by the `RecommendationsFactory`.

Search Strategies:

- Provide functionality to search for books using different approaches (exact matching or approximate search).
- Share a common interface (`SearchStrategy`) with a template (`SearchStrategyTemplate`) to standardize search behavior.
- Created by the `SearchFactory`.

Factories (`RecommendationsFactory`, `SearchFactory`):

- Responsible for the instantiation and management of respective strategies.
- Centralize strategy creation to support a clean, modular architecture.