Online Social Book Store Application

DESIGN, IMPLEMENTATION, INSTALLATION, BUILD & EXECUTION GUIDES

Table of Contents

1 Introduction

This document provides some basic but important guidelines for the development of the project.

2 Application Design and Related Design Patterns

2.1 Architecture

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**. Maintainability and extensibility are further promoted by the fact that the **Spring Boot framework** heavily relies on the **Model View Controller (MVC) pattern** for the development of Web applications. MVC allows the clear separation of three different concerns: the views of the application that areresponsible 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 controllers that take user input, manipulate the domain model data and update the views.

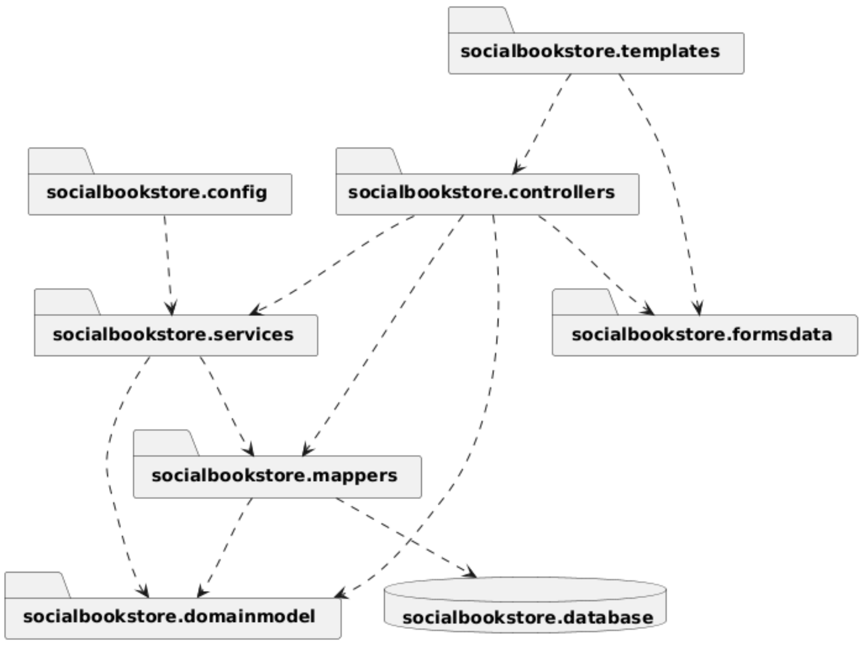
socialbookstore.templates
socialbookstore.config
socialbookstore.controllers
socialbookstore.services
socialbookstore.formsdata
socialbookstore.mappers
socialbookstore.domainmodel
socialbookstore.database

FIGURE 1 APPLICATION ARCHITECTURE.

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

- The **templates package** realizes the user interaction (UI) with the application in collaboration with the controllers. 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 are responsible for passing the appropriate domain model objects to the view layer. We can think of views as html templates consisting of html code and “special” expressions (${...}) for accessing domain model data, performing calculations, and controlling the display of content. These expressions are processed by the so-called view **template engine** that executes the “special expressions” and consequently incorporates the results in the html code and produces a complete html page, which is visualized in the end user Web browser. In general, there are several alternative template engines that can be used to develop the front end of a Spring application. Our application is based on **Thymeleaf**.

- The **controllers package** consists of controller classes which take user input from the views of the application, perform certain **operations** 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 **domain 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.

- Currently, the **services package** is responsible for user authentication and authorization via respective registration and login processes. These processes are designed and implemented with respect to the **Spring Security framework** that is part of the overall Spring Boot framework.

- The **mappers package** is responsible for basic database operations which map the domain model data, stored in the database tables 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**. Moreover, the implementation of this package is designed and implemented with respect to the **Spring Data JPA**.

- 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.

2.2 Domain Model

User and **Role** are specific classes we implement for the realization of the user registration and login actions in **Spring Boot**. User should implement the Spring **UserDetails** interface for this reason.

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 favoriteBookAuthors and a list of favoriteBookCategories. 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 [1].

For the realization of the book offers search and recommendation we rely on the GoF **strategy pattern**. The basic reason for these design choice is the **maintainability requirements** given in the application requirements document for the **easy extension of the application with new strategies** in the future. The search and recommendations strategies are implemented in the **domainmodel.searchstrategies** and the **domainmodel.recommstrategies** subpackages of the **domainmodel** package. Specifically, for the search strategies the idea is to define a common SearchStrategy interface for the different search strategies (see GoF strategy pattern). The search algorithms are implemented by classes that implement the SearchStrategy interface. The design of the book offer recommendations strategies is similar.

2.3 Data Mappers

The mappers package consists of several interface definitions, derived from the general **JpaRepository** interface that is provided by **Spring Boot Data JPA**. Specifically, the contents of the package comprise the UserProfileMapper, the UserMapper, the BookMapper, the BookCategoryMapper and the BookAuthorMapper.

2.4 Services

This package comprises a service that is responsible for the user registration and login. The service provides the UserService interface that is implemented by the UserServiceImpl class. The service further implements the UserDetailsService interface, as required by the **Spring Security framework**.

2.5 MVC- Controllers, Template 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 remaining user stories. The dynamic html views contained in the **templates package** are based on the **Thymeleaf** template engine.

1 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

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.

2.6 Configuration

The classes in the configuration package are responsible for the configuration of the Spring Boot framework and the configuration of the Spring Security framework.

3 Installation, Build and Execution of the Application

The online social bookstore is a **Spring Boot** application developed in **Java (v17)**.

1. The source code is packaged as a **Maven project**. Maven is an automated application management and build tool. Maven uses an XML file called **pom.xml** (Project Object Model) to manage the project's configuration, dependencies, plugins, and more. The project dependencies (<dependency> …. </dependency>) specify required APIs which are automatically downloaded by Maven and linked with the project. As it is typically done in Spring Boot applications, the initial version of the project has been created using the **Spring Boot initializr** (https://start.spring.io/) tool that creates the project structure and pom.xml. The automatic build process of the project results in a jar file that contains an embedded application server (**Tomcat** in particular) that is used for the execution of the project (hence there is no need to install Tomcat to run the application). The application server is responsible for listening and handling the http requests that are targeted to the application. Embedding the application server in the final jar file is a feature that has been specified when the project has been created using the initilzr tool.

2. To install the application, you must import it as an existing Maven project in the Eclipse IDE (using other IDEs is also possible via similar stpes). From the **Eclipse** File menu go to **File>Import>Maven>Import Existing Maven Projects** and select to project folder (socialbookstore) that is provided to you as part of the project material.

3. The project relies on a **MySQL** database for storing and retrieving data such as the user profiles, book offers, book requests and so on. The sql script that defines the relational schema that is assumed for the project data is located in **socialbookstore/myy803_socialbookstore-static.sql**. Therefore, to execute the project you need a **MySQL installation**. Then, you have to create the database by executing the aforementioned script. Finally, you **must configure the connection to**

the database. To this end, you have to modify the socialbookstore/src/main/resources/**application.properties** file. Specifically, you must modify the values of the **JDBC properties** in this file with appropriate values that correspond to the username, password of your database connection and the URI of the database that you created.

4. To execute the application, you can simply right click on the **SocialbookstoreApplication** class and select **Run As>Java Application**. In its current state the application is configured to run on the localhost:8080. So, when the application context is set and the application up and running you can start using the application via a web browser by visiting the localhost:8080 URI.

4 TODO STUFF

4.1 Tests

The test base of the application is EMPTY in this preliminary version. Ideally, 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** (for some examples see this git repository).

4.2 Unimplemented User Stories

The user stories that concern the deletion of book offers and requests are not implemented in this version. The notification of the users about the outcome of their book requests is also not fully implemented.

5 APPENDIX Spring Boot Annotations

Spring Boot and its sub-frameworks (Security, Data JPA, etc.) heavily rely on annotations for the specification of the different application components and the binding between these components. A quick guide of the main annotations that can help understanding the source code of the application can be found here.