Software Project BookHub

Cedric Bürkel, Jonas Bujok
February 15, 2025

1 Introduction of the Project Topic

The software project named BookHub is a Java application based on Spring, which enables the users to manage and retrieve informational data about books, authors, publisher, genres, reviews and the users themselves. The main objective of this application was to create a platform for people who are passionate about books, in which they could find all relevant information and meta data. Besides the simple retrieval of information users can add new records, update existing ones or delete them. Some typical use cases would be to find books based on an author or adding a new author and some books he has written.

2 Description of the Software Architecture

The project is built upon the Hexagonal Architecture, which provides a clear decoupling between the different layers of the application. This architecture enables a better maintainability and testability. This is especially an advantage in a dynamic development environment.

2.1 Domain

The domain layer covers the fundamental business objects and the core logic of the application. In this project these are the model classes like 'Autor' or 'Bewertung', which define the essential characteristic of those objects and represent the relationships to other business objects. Each model contains the necessary attributes, e. g. 'vorname', 'nachname', 'geburtsdatum' and 'nationalität' for an 'Autor' and creates creates the basis for the entire business related logic.

2.2 Business Logic

The business logic is implemented in the service classes, e. g. 'AutorService' or 'BewertungService'. This layer fulfils a central role and covers the implementation of business rules and business processes. The class 'AutorService' manages all interactions with authors like creating, updating and deleting. This way the business logic stays separated of the technical details of the data processing and the user interface.

The design of the service classes concentrates on maintaining integrity of the business processes. For instance, the class 'AutorService' ensures that all relevant author data is checked and correctly saved and that the relationships between books and authors remain consistent. The use of Java annotations like '@Cacheable' and '@CacheEvict' within the services improves performance due to efficient database queries and caching strategies.

2.3 Infrastructure Components

The infrastructure of the application forms the repository layer, which manages access to the database. This is were the, e. g. 'AutorRepository', comes into play, which is based on the Spring Data JPA and efficiently implements CRUD operations. The repositories encapsulate the data access technology, by which the rest of the application remains independent of the way data becomes persistent.

The Hexagonal Architecture promotes use of interfaces, which helped us to decouple the independence of the application of the specific implementations. The repositories act as interfaces between services and the the physical database, where queries to abstractions of database operations are defined by JPA method signatures.

2.4 Ports and Adapter

In the context of the Hexagonal Architecture, the ports operate as interfaces, which are defined by the application logic and state which functionalities have to be made available. Adapters, on the other hand, implement those interfaces and facilitate communication with the outside world.

The 'AutorController' class serves as example for a primary adapter in this project. The controller operates as HTTP interface and is responsible for translating http queries to serviceable method calls. Every endpoint in a controller is carefully planned out to provide an intuitive and REST compliant API. Beyond that we use classes like 'AutorModelAssembler' as secondary adapter, which ensures that the hypermedia principle or HATEOAS is preserved.

A crucial aspect in learning to implement the Hexagonal Architecture was the challenge to maintain the consistency between layers while

preserving loose coupling. This architecture required careful planning and knowledge of the design principles to ensure that the application remains efficient, flexible and extensible. By the use of a modular approach, we could implement tests easier, which improved the reliability of the system overall.

3 Explanation of the API Technology

Out RESTful API technology is based on the Spring Web Framework. The choice was made due to it being strongly integrated into the Spring environment and having a great community, which already solved most of the problems we could have get in implementing such an application. REST enables to use standardized HTTP queries with GET, POST, PUT and DELETE, which is commonly used in many other applications over the web and easy to implement.

The main advantages of using REST in this project are the simple URL structuring and the variety existing tools for documentation and testing purposes. A trade-off lies in the complexity of implementing security and further extensions, but this is well supported by the Spring ecosystem.

4 Implementation Details

4.1 Authentication

The implementation includes no advanced security measures as this point, but could be implemented in the future.

4.2 Adapter Mapping

The use of assembler classes like 'AutorModelAssembler' clarifies the transformation of the models in API suitable representations with the help of Spring HATEOAS implementations.

4.3 Choosing a framework

The decision to use Spring was based on the requirement for a robust, extensible platform with good support for data access, web based development and testing. Spring offers a reliable infrastructure, which can be adjusted for many different use cases.

5 Testing Strategy

Our testing strategy follows a structured approach, which covers unit tests as well as integration tests. This strategy was developed to ensure functionality and stability of the application while reaching an extensive test coverage.

5.1 Unit Tests

Unit tests are a central component of our test strategy. They are being used for certain parts of the business logic, especially for testing the methods of the service classes for correct functionality. By using Mockito we can mock dependencies and therefore test the services isolated. For example, when testing 'AutorService', the interaction with the 'AutorRepository' is simulated. Such tests allow us to test methods like 'addAutor()' and 'updateAutor()' in isolated scenarios, which ensures that they work correctly and independent of other components.

Another focus point of unit tests lies in validation of the business logic itself. For example, it is checked if all necessary fields are set correctly. Unit tests make it easier to identify faulty logic and fix these in an early stage.

5.2 Integration Tests

Integration tests make sure that all components of the application work together seamlessly and correctly. We are using Spring Boot Test Framework in our project to load the entire application stack, containing the web layer, database access and services. These tests check the entire application logic end-to-end by simulating HTTP requests using MockMvc and validating the response codes as well as the return values.

Special attention has been given to how the application responds to typical user interactions, like creating, reading, updating and deleting of authors, books and other objects based on the model classes. The integration tests also identify compatibility issues between components that may not be detected during isolated testing.

6 Learning Outcomes and Reflection

While developing the BookHub project, we have learned many things about structured software development, especially in Spring, and the advantages of Hexagonal Architecture. Particularly valuable has been the realization of flexibility of this architecture in supporting tests and implementation of new features. It was tricky initially to get used to this new structured kind of developing, but with time it made click and we knew how to handle this approach.