Bookshop Management System

: Assignment 1

Student: Grad Laurentiu-Calin

**Group: 30435**

Table of Contents

Contents

[1. Requirements Analysis 3](#_Toc193146482)

[1.1 Assignment Specification 3](#_Toc193146483)

[1.2 Functional Requirements 3](#_Toc193146484)

[1.3 Non-functional Requirements 3](#_Toc193146485)

[2. Use-Case Model 3](#_Toc193146486)

[3. System Architectural Design 4](#_Toc193146487)

[3.1 Architectural Pattern Description 4](#_Toc193146488)

[3.2 Diagrams 5](#_Toc193146489)

[4. Class Design 6](#_Toc193146490)

[4.1 Package + Class Diagram 6](#_Toc193146491)

[5. Data Model 6](#_Toc193146492)

[6. Bibliography 6](#_Toc193146493)

# 1. Requirements Analysis

## Assignment Specification

The aim of this project is to develop my aptitudes in the field of software engineering, especially in the areas of system design and project management, and to become familiar with widely used programming paradigms, programming languages and frameworks. For this purpose, I chose to implement a well-known project: a bookstore management system. Even though the theme is rather simple, the focus of the project is to understand the capabilities of the technologies I will work with.

## Functional Requirements

There are three main “things” users should be able to interact with: authors, books and publishers. These are the foundations of the functional requirements:

1. Allow users to read, update and delete old entries or create new entries for **publishers** by specifying: the name, the location of their headquarters and the year of foundation.
2. Allow users to read, update and delete old entries or create new entries for **authors** by specifying: the first and last name, an alias (optional) and the nationality.
3. Allow users to read, update and delete old entries or create new entries for **books** by specifying: the ISBN, the title, the authors, the publisher, the year of publication, the price and the available stock.
4. Allow admins to read, update and delete existing users of the system.
5. Allow users to register in the system by specifying: a username, a password, the first and last name and their role.
6. Allow users to login in the system.
7. Allow users to add books to their virtual cart and place orders.
8. Allow users to sort items by category.

## Non-functional Requirements

Used technologies:

* Database management: PostgreSQL
* Backend development: Java & SpringBoot
* Frontend development: React & Vite

Some of the non-functional requirements:

1. Implement validation to ensure all inputs adhere to a specified format (e.g. ISBN is a 10/13-digit code) and thus keep the store information consistent.
2. Impose only highly-complexity passwords (e.g. containing at least a capital, a digit and some special symbols).
3. Use an ORM to handle database interaction.
4. Use a DI container for dependency injection.
5. Implement layered architecture for better organization and separation of concerns.
6. Use a SQL database to store required information.

# Use-Case Model

A screenshot of a black background

AI-generated content may be incorrect.

Figure . The uses cases of the bookstore management system.

## Description of the User role:

Note: Any employee or administrator is a user.

|  |  |  |  |
| --- | --- | --- | --- |
| Use case | Description | Main success scenario | Extensions |
| Register | The user fills in a form with some personal details and is registered by our system, from now on it can log into his personal account. | The database records a new user of the system. | Fields that do not comply with the specified standard generate errors that are displayed in the GUI. |
| Login | A pre-registered user accesses his account. | Successful identification based on the provided credentials. | Unsuccessful/Successful logins are signaled in the GUI. |
| Fetch all entities\* | Fetching all the entries in the database that correspond to an entity and displaying them in a table. | Backend and database connection succeed, and the entities are displayed in an intuitive manner. | Any errors that happen during the fetching are reported in the GUI, as well as information messages (“No entities\* are present!”) |
| Add to cart | *TODO* |  |  |
| Pay | *TODO* |  |  |

(**\***entity = book | author | publisher)

## Description of the Employee role:

Note: Administrators are employees.

|  |  |  |  |
| --- | --- | --- | --- |
| Use case | Description | Main success scenario | Extensions |
| Insert a new entity\* | Register in the database a new entity instance. | The database records a new instance of the entity type. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |
| Update an existing entity\* | Update an existing instance from the database. | The database updates the existing instance of the entity type. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |
| Delete entities\* | Delete one or more entity instances from the database. | The database drops the specified (existing) instances of the entity type. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |

(**\***entity = book | author | publisher)

## Description of the Admin role:

|  |  |  |  |
| --- | --- | --- | --- |
| Use case | Description | Main success scenario | Extensions |
| Fetch all users | Fetching all the user entries in the database and displaying them in a table. | The user entries are displayed in a table | Any errors that happen during the fetching are reported in the GUI. There should always be at least one admin (user) of the system. |
| Insert a new user | Register in the database a new user instance. | The database records a new user with the specified credentials. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |
| Update an existing user | Update an existing user from the database. | The database updates the existing user. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |
| Delete users | Delete one or more users from the database. | The database drops the specified (existing) users. | Any errors that appear at the database level (unique constraint violation error) or at the logic layer (validation errors) are reported in the GUI. |

# System Architectural Design

## Architectural Pattern Description

The system is designed based on the three-server architectural pattern: the data tier, the logic tier and the presentation tier are run concurrently. The theoretical advantages of this strategy are portability – tiers can be deployed on different platforms without major changes; scalability – each tier scales independently of the others; maintainability – updates in one tier do not affect the structure of the others; security – each layer can integrate different layers of security, thus restricting the access to sensitive data; reusability – usually, the business logic can be reused across several systems; flexibility – different technologies and frameworks can be used for each tier.

A diagram of a server

AI-generated content may be incorrect.

Describe what implementation you used for each tier.

## Diagrams

And architecture should be layered:

**Bar chart

Description automatically generated with low confidence**

Make a diagram with YOUR OWN layers and say what they are and why.

# Class Design

## Package + Class Diagram

Diagram

Description automatically generated

The diagram should contain the modules and high level classes, as seen above.

# Data Model

Describe your data model, alongside a **diagram of the database (entity – relationship**). You can probably generate it straight from IntelliJ.

# Bibliography

1. <https://www.npmjs.com/package/react-multi-select-component>
2. <https://www.npmjs.com/package/react-data-table-component/v/7.0.0-rc2>
3. <https://medium.com/trendyol-tech/how-to-write-a-spring-boot-library-project-7064e831b63b>
4. <https://react.dev/learn/thinking-in-react>
5. <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
6. <https://www.dhiwise.com/post/exploring-react-datatable-everything-you-need-to-know>