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 add books to their virtual cart and place orders.
7. Allow users to sort items by category.

## Non-functional Requirements

Used technologies:

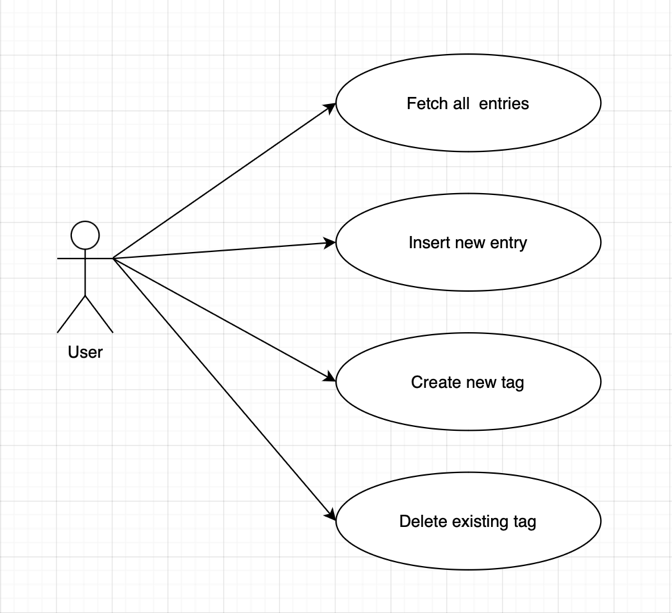
* Database management: PostgreSQL
* Backend development: Java & SpringBoot
* Frontend development: React & Vite

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. Use an ORM to handle database interaction.
3. Use a DI container for dependency injection.
4. Implement layered architecture for better organization and separation of concerns.
5. Use a SQL database to store required information.

# Use-Case Model

Example content:



Description of the “Fetch all entries” use-case:

Use case goal: Fetching all of the recent entries and displaying them on screen using a graph representation

Primaray actor: The user of the application

Main succes scenario: Database connection succeeds, data is fetched correctly and is displayed in an intuitive manner within graphs

Extensions:

* Alternate scenario of success: Database connection succeeds, data is fetched correctly, but no entries are contained in the response due to the user never having inserted one, the home screen displays a message suggesting that user should insert his first entry
* Failure: Either the database connection fails, the request times out, the data is corrupt, the graph is unreadable or does not populate with data or the Android app crashes unexpectedly. The user will get a corresponding message in the frontend.

# 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>