Your Books Everywhere!

Analysis and Design Document

Student: Ciuca Daniel Claudiu

**Group: 30238**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The task is to build a book management service. The service helps users to borrow or return a book without having to go to a library. It’s very useful for the people that don’t have time to go to a library or for example if it’s raining out.

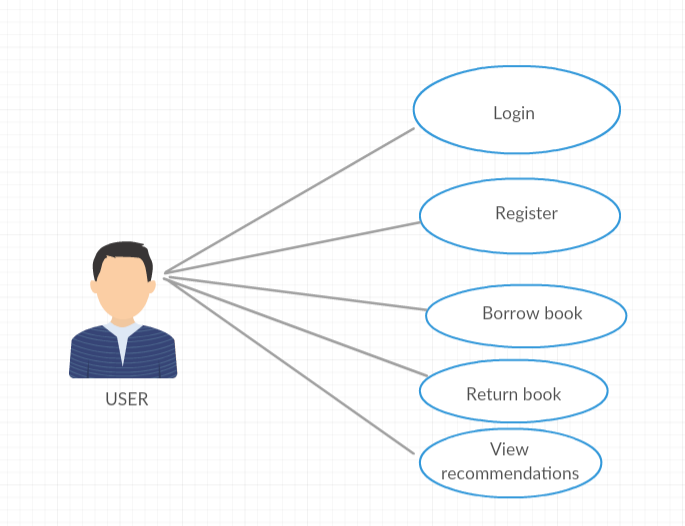
# Functional Requirements

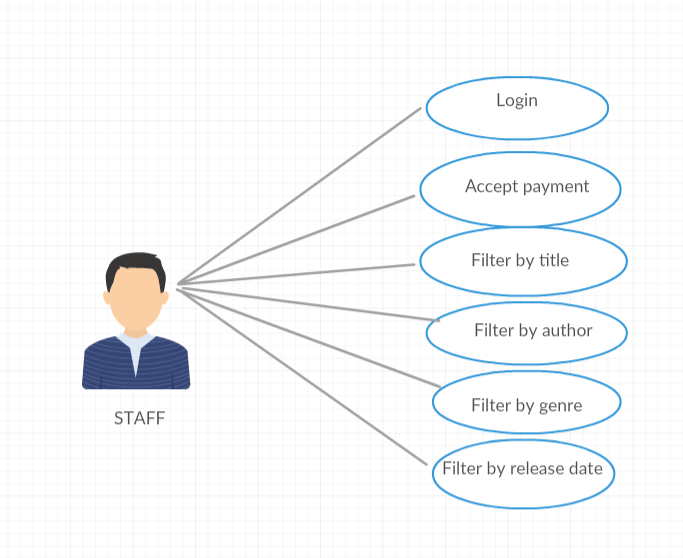
* Account creation: a user can create an account and have to choose username, password and payment method (Day, Week, Month, Year).
* Staff management: the staff can to accept a user (validate payment), filter books by title, author, genre and release date.
* Borrow a book: if a book is available a user can add it to your library. If not the user can join a waiting list.
* Return a book: Once a book has been read by a user it can be returned via the online library return function. This assigns the book to the next user in the waiting list after validation of the return by library staff.
* Recommendation service: The app provides recommendations to user by trend, user interest by genre.

# Non-functional Requirements

* Implement the application and test it
* Use an OOP language (Java)
* Use a CQRS architecture, use a mediator pattern to handle requests
* Use a decorator pattern for applying ownership behavior to a book (borrowed, unavailable)

2. Use-Case Model





Use case: Borrow a book

Level: user-goal level

Primary actor: User

Main success scenario:

* User clicks Register for the menu
* User introduces username, password and payment plan
* Staff accept user
* User click Back to login page
* User introduces username and password
* User clicks Login
* User introduces book’s title
* User clicks Borrow

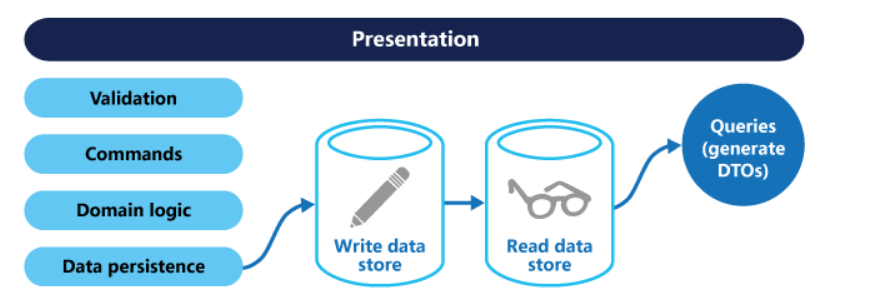
Extensions:

* Staff don’t accept user payment plan

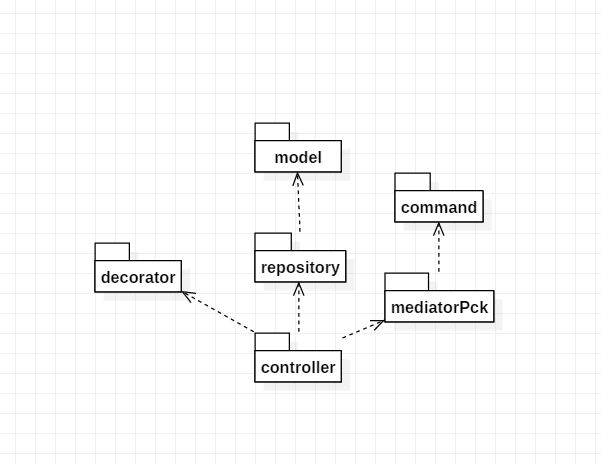
3. System Architectural Design

**3.1 CQRS Description**

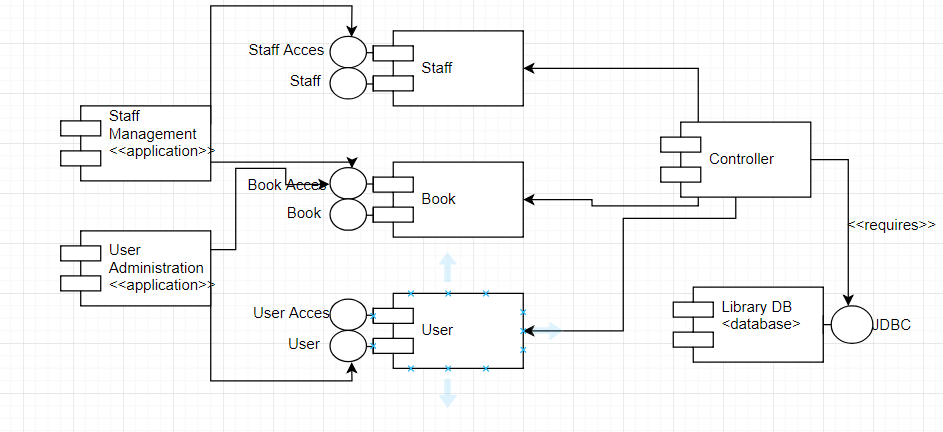
Segregate operations that read data from operations that update data by using separate interfaces. This can maximize performance, scalability, and security. Supports the evolution of the system over time through higher flexibility, and prevents update commands from causing merge conflicts at the domain level.



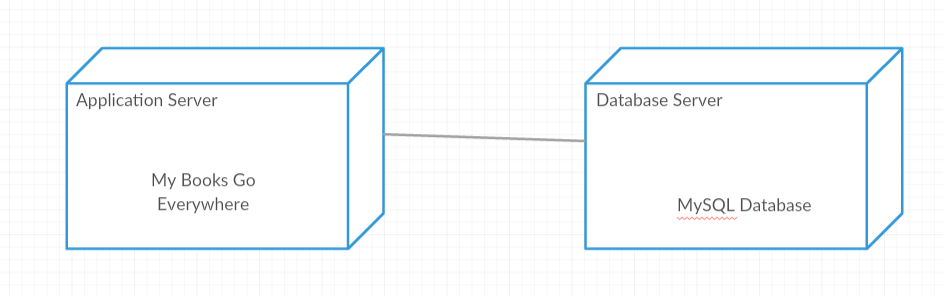
**3.2 Diagrams**



Package Diagram

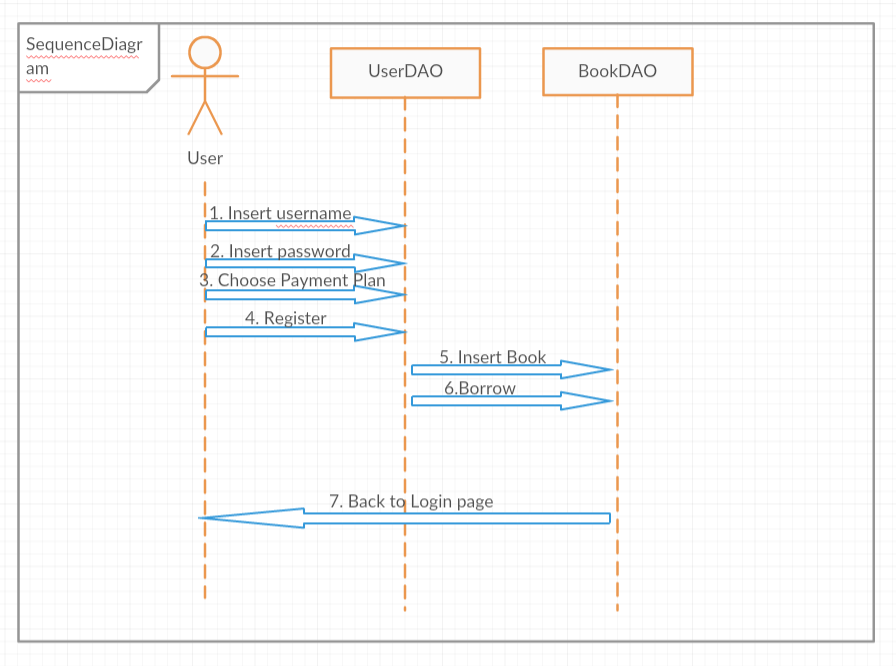


Component diagram



Deployment diagram

4. UML Sequence Diagrams

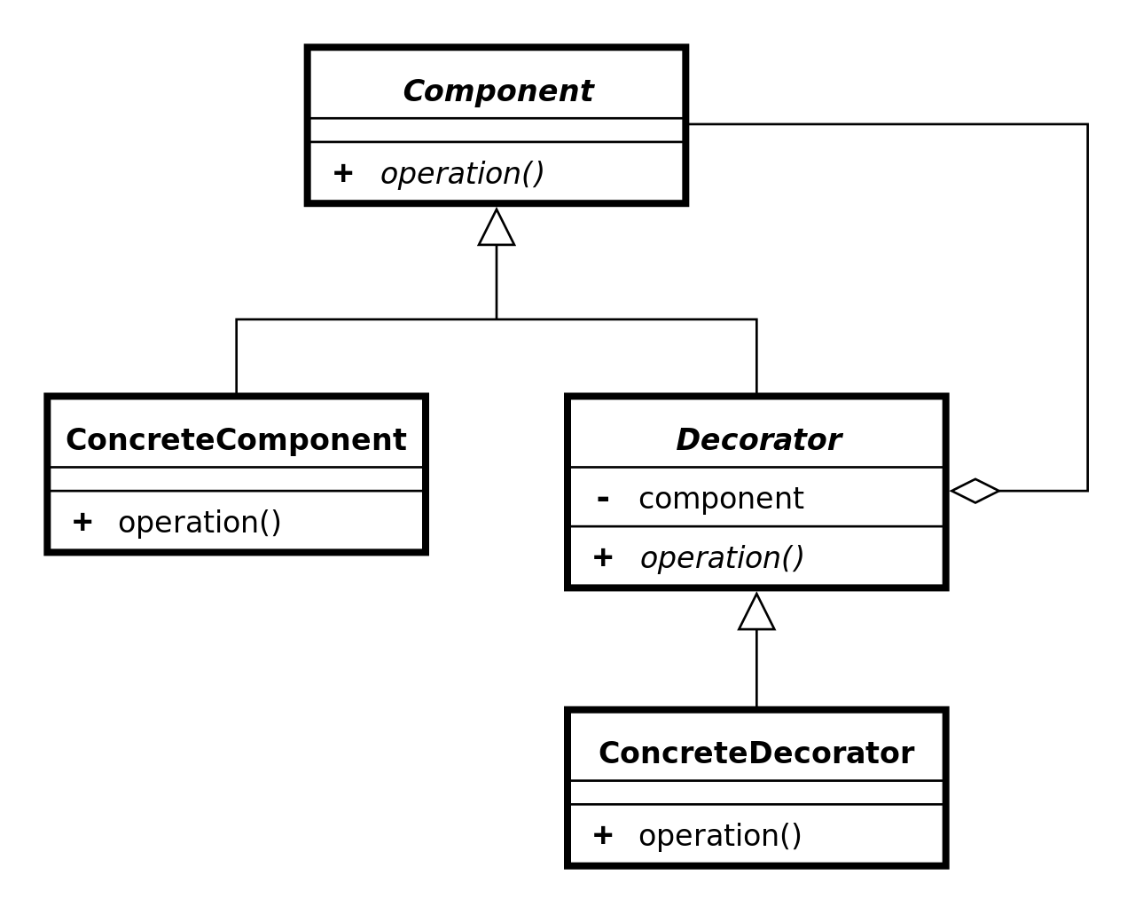


5. Class Design

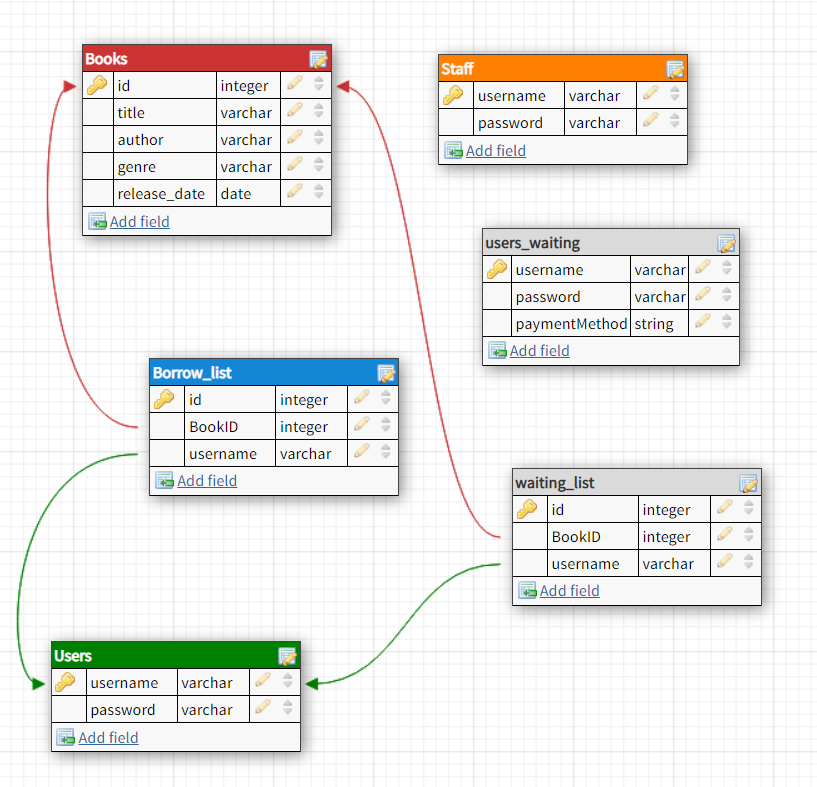
**5.1 Design Patterns Description**

In object-oriented programming, the decorator pattern is a design pattern that allows behavior to be added to an individual object, dynamically, without affecting the behavior of other objects from the same class. The decorator pattern is often useful for adhering to the Single Responsibility Principle, as it allows functionality to be divided between classes with unique areas of concern. The decorator pattern is structurally nearly identical to the chain of responsibility pattern, the difference being that in a chain of responsibility, exactly one of the classes handles the request, while for the decorator, all classes handle the request.

Example:



6. Data Model

7. System Testing

To test my application, I create a DataJpaTest class that do 2 tests:

* In first, it verifies the registration of a user.
* In second, it verifies if a user is accepted by a staff

8. Bibliography

Decorator Pattern:

<https://en.wikipedia.org/wiki/Decorator_pattern>

CQRS Architecture:

<https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>

Sequence diagram:

<https://en.wikipedia.org/wiki/Sequence_diagram>