Your Books Everywhere!

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.

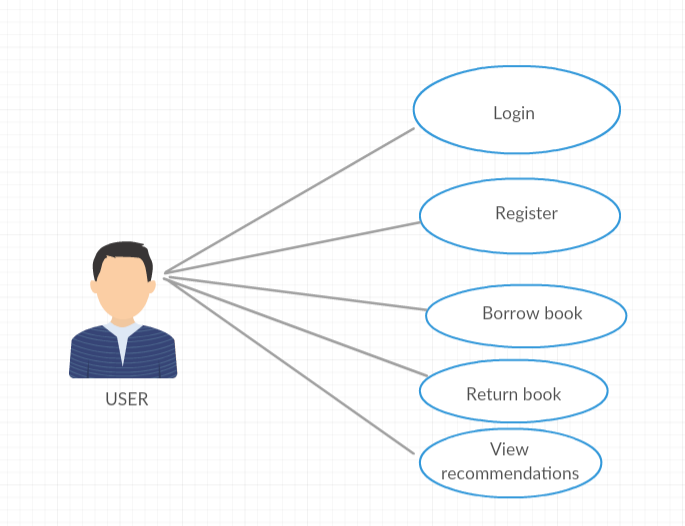
# Functional Requirements

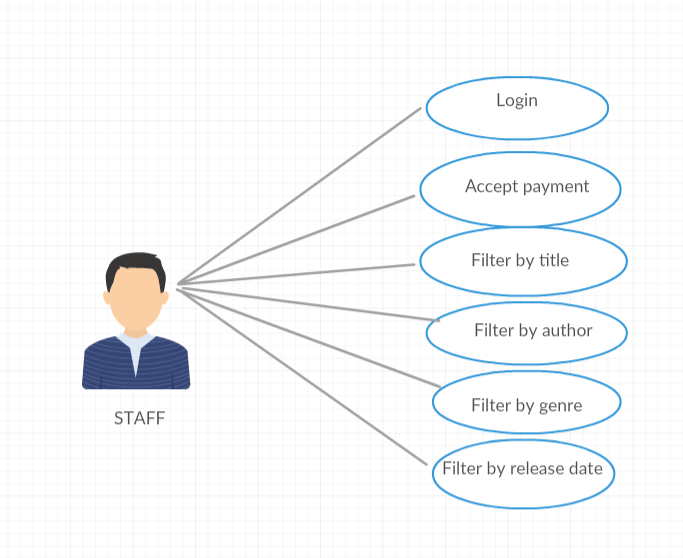
* Account creation: a user can create an account and have to choose username, password and payment method (Day, Week, Month, Year).
* Staff management: the staff can to accept a user (validate payment), filter books by title, author, genre and release date.
* Borrow a book: if a book is available a user 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 client-server architecture
* Use an observer for getting notifications when a book becomes available

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

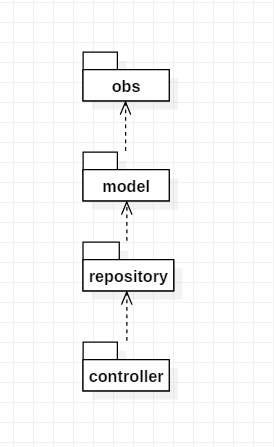
**3.1 Client-Server Description**

Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function.

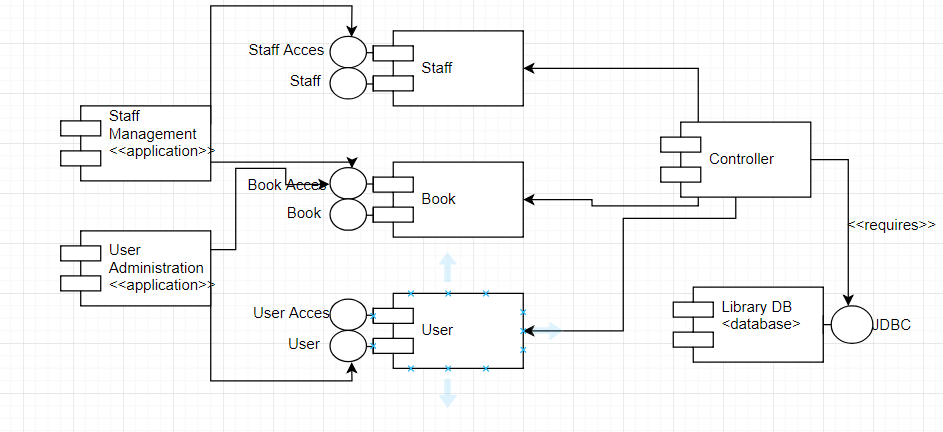
O imagine care conține captură de ecran

Descriere generată automat

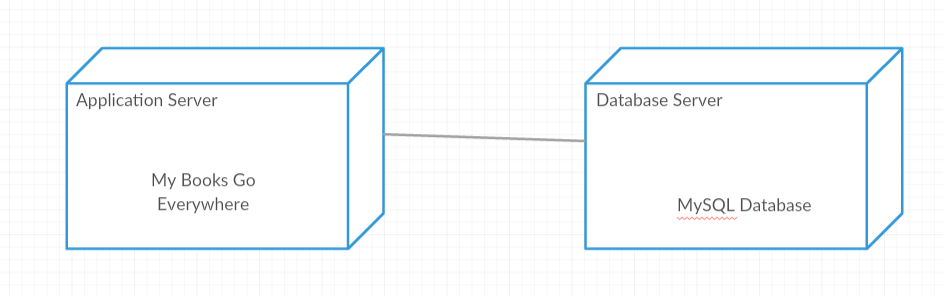
**3.2 Diagrams**



Package Diagram

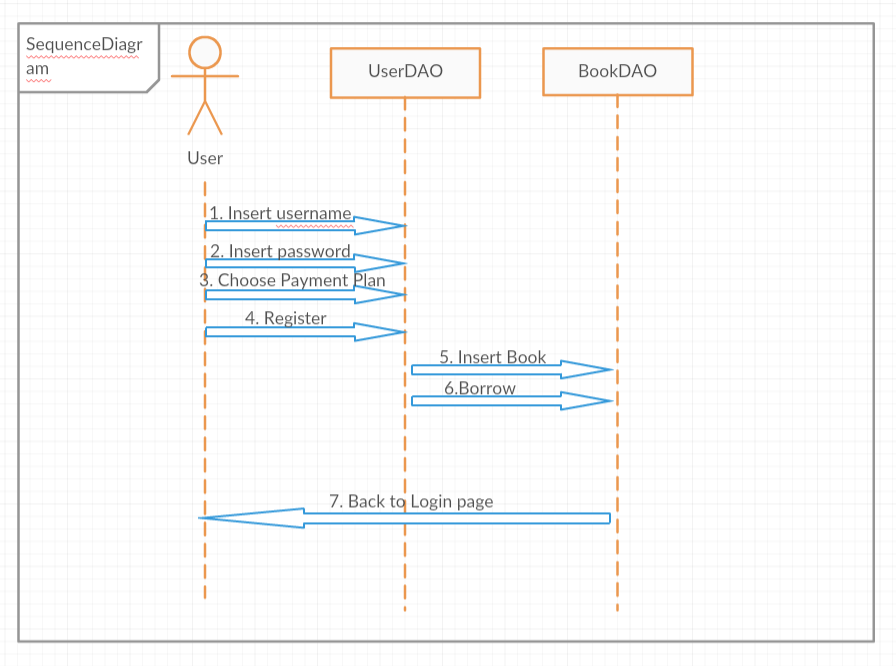


Component diagram



Deployment diagram

4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

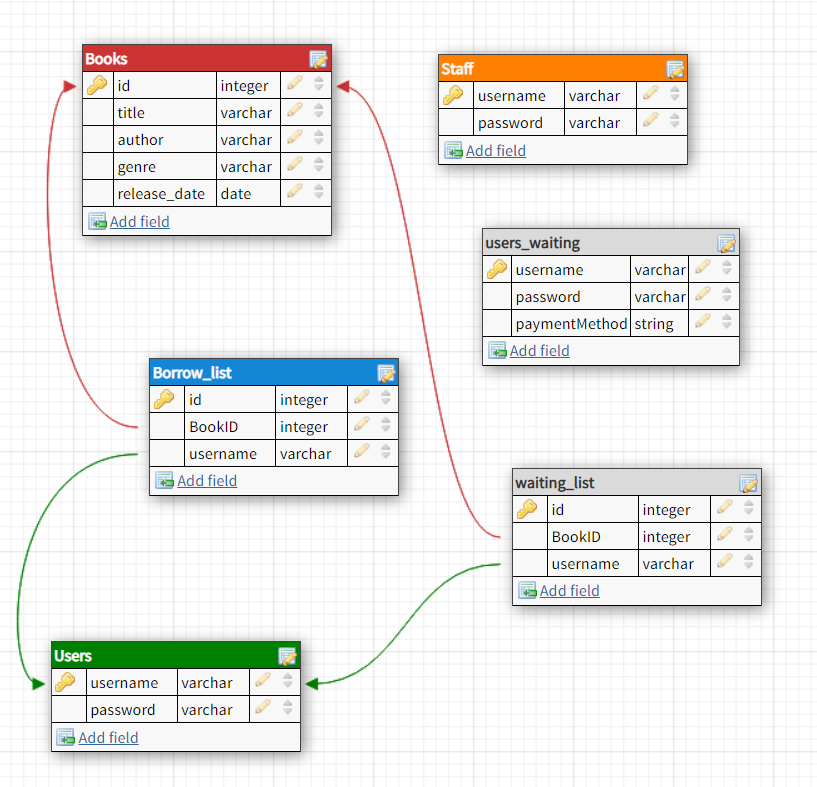
It is mainly used to implement distributed event handling systems, in "event driven" software. Most modern languages such as C# have built-in "event" constructs which implement the observer pattern components.

Example:

O imagine care conține captură de ecran

Descriere generată automat

6. Data Model

7. System Testing

To test my application, I create a JUnit class that do 3 tests:

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

8. Bibliography

Observer Pattern:

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

Client-Server Architecture:

<https://en.wikipedia.org/wiki/Client%E2%80%93server_model>

Sequence diagram:

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