Your Books Everywhere!

Student: Alexandra Georgeana Czako

**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

You are tasked to build a book management service.

A user should be able to create an account, choose a payment plan and login to search the book library.

Payments can be done via a cash only policy and need to be validated by library staff.

The library is managed by staff and can be filtered by release date, author, title, genre.

If a book is available a user can add it to your library. If not the user can join a waiting list. 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.

The service also provides users with dynamic recommendations based on latest trends (popular borrowed books) or user defined interests by genre or topic

# Functional Requirements

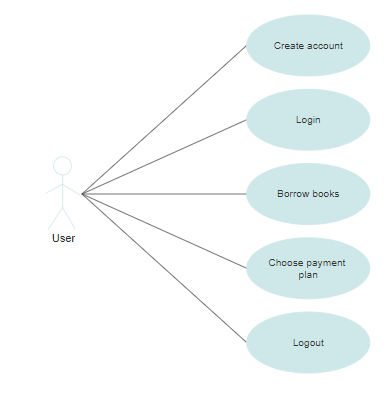
* Create account for user
* Choose payment plan for user
* Filter all the books by: author, title, genre; Show all the books available in the library
* Validate payments by staff

# Non-functional Requirements

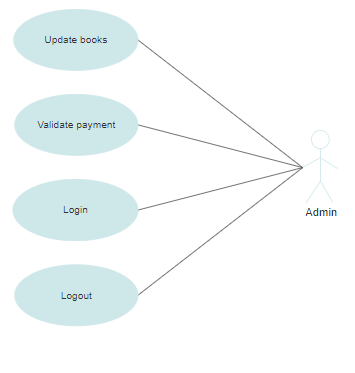
* Secured password
* Junit tests

2. Use-Case Model

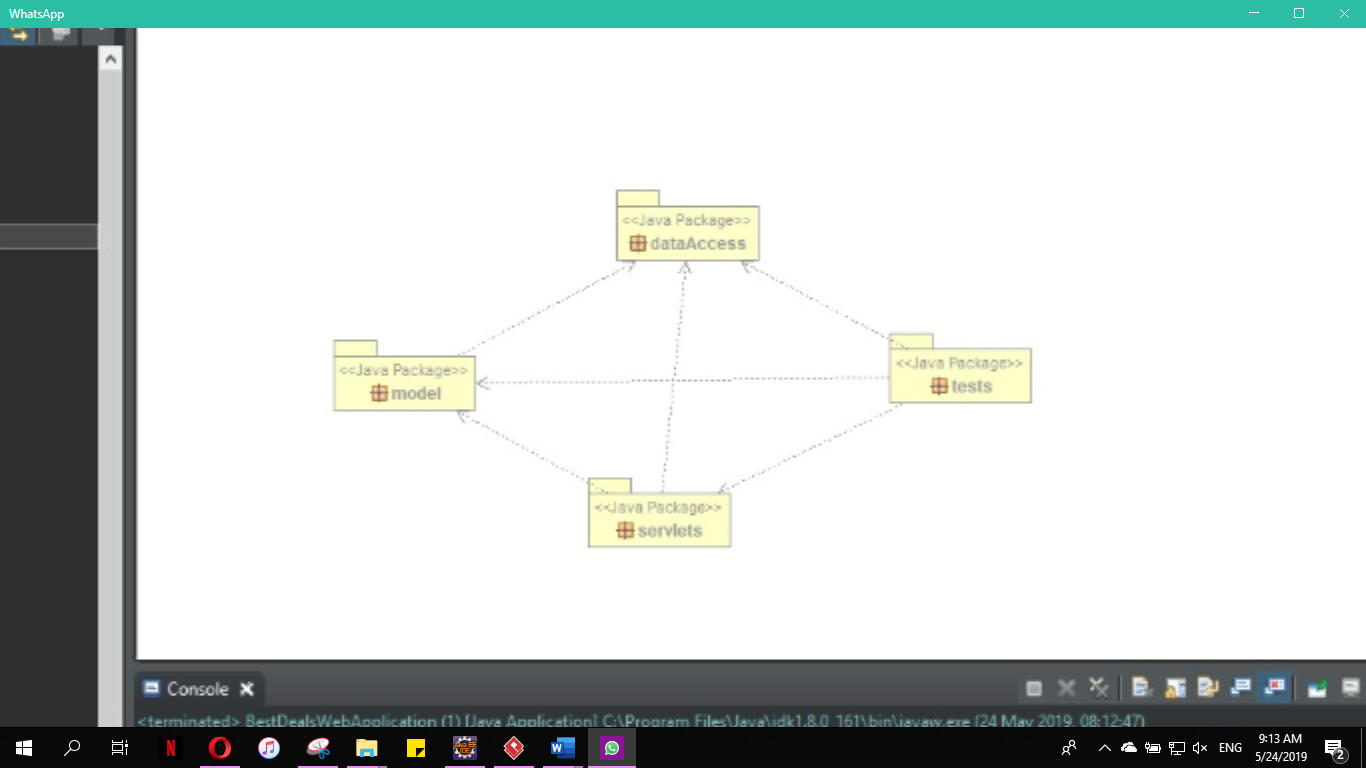
User’s use case model:



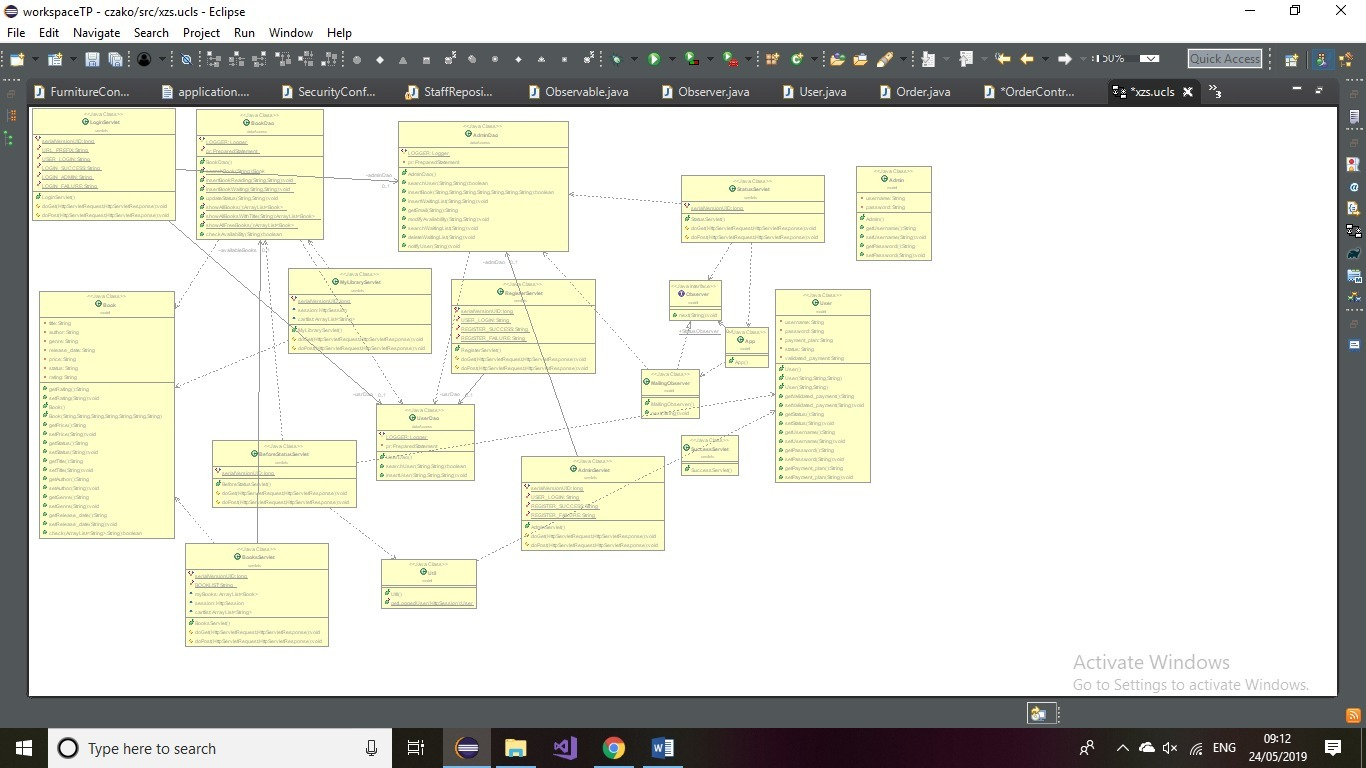
Admin’s use case model:



**Package diagram**



**Class diagram**



3. System Architectural Design

**3.1 Architectural Pattern Description**

**Client–server architecture** is a [distributed application](https://en.wikipedia.org/wiki/Distributed_application) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](https://en.wikipedia.org/wiki/Server_(computing)), and service requesters, called [clients](https://en.wikipedia.org/wiki/Client_(computing)). Often clients and servers communicate over a [computer network](https://en.wikipedia.org/wiki/Computer_network) on separate hardware, but both client and server may reside in the same system. A server [host](https://en.wikipedia.org/wiki/Host_(network)) 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. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are [Email](https://en.wikipedia.org/wiki/Email), [network printing](https://en.wikipedia.org/wiki/Network_printing), and the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web).

5. Class Design

**5.1 Design Patterns Description**

The **observer pattern** is a [software design pattern](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) in which an [object](https://en.wikipedia.org/wiki/Object_(computer_science)#Objects_in_object-oriented_programming), 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](https://en.wikipedia.org/wiki/Method_(computer_science)).

### **What problems can the Observer design pattern solve?**

The Observer pattern addresses the following problems:

* A one-to-many dependency between objects should be defined without making the objects tightly coupled.
* It should be ensured that when one object changes state an open-ended number of dependent objects are updated automatically.
* It should be possible that one object can notify an open-ended number of other objects.

Defining a one-to-many dependency between objects by defining one object (subject) that updates the state of dependent objects directly is inflexible because it couples the subject to particular dependent objects. Tightly coupled objects are hard to implement, change, test, and reuse because they refer to and know about (how to update) many different objects with different interfaces.

8. Bibliography

* <https://www.sciencedirect.com/topics/computer-science/business-logic-layer>
* <https://objcsharp.wordpress.com/2013/07/22/what-is-a-business-logic-layer-anyway/>
* <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>