Your Books Everywhere!

Analysis and Design Document

Student: Bianca Elena Dondas

**Group: 30238**

Contents

[1. Requirements Analysis 3](#_Toc4878772)

[1.1 Assignment Specification 3](#_Toc4878773)

[1.2 Functional Requirements 3](#_Toc4878774)

[1.3 Non-functional Requirement 3](#_Toc4878775)

[2. Use-Case Model 3](#_Toc4878776)

[3. System Architectural Design 4](#_Toc4878777)

[3.1 Architectural Pattern Description 4](#_Toc4878778)

[3.2 Diagrams 5](#_Toc4878779)

[4. UML Sequence Diagrams 5](#_Toc4878780)

[5. Class Design 6](#_Toc4878781)

[5.1 Design Patterns Description 6](#_Toc4878782)

[5.2 UML Class Diagram 7](#_Toc4878783)

[6. Data Model 8](#_Toc4878784)

[7. System Testing 8](#_Toc4878785)

[8. Bibliography 8](#_Toc4878786)

# 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

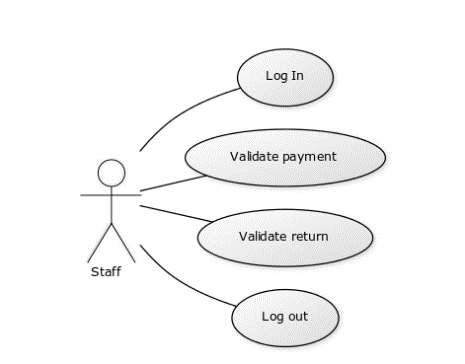
## Functional Requirements

* Log in and create account
* Choose payment plan
* Filter by: author, title, genre
* Borrow & return service
* Validate payments and validate return book by staff

## Non-functional Requirement

* Using OOP language (Java)
* Commit work to GitHub
* Client-Server architecture
* Store data in MySql Database
* Use observer pattern

# 2. Use-Case Model

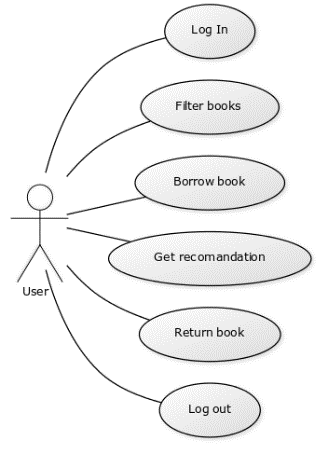


Use case: Validate return

Level: User- goal level

Primary actor: Staff

Main success scenario: The librarian connects to the app using the log in page. Then he check the return requests. He remove the person from return request table and the assign the book to the next person from the reading list.



Use case: Borrow book and return

Level: User- goal level

Primary actor: User

Main success scenario: The user connects to the app using the log in page. Then he can choose a book from the list. When he decides what he wants to read, he adds the book to his reading list. Once he finish the book he request the return of the book, which will be validated by staff.

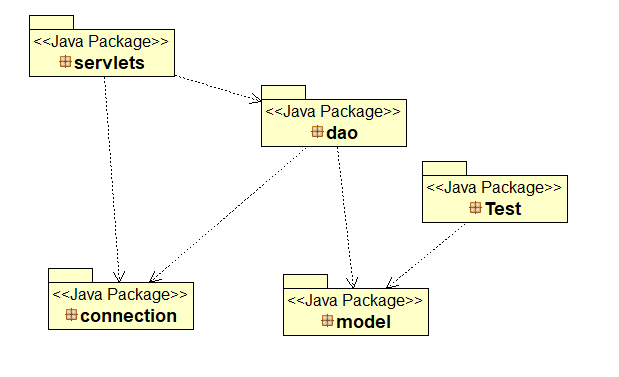
# 3. System Architectural Design

## 3.1 Architectural Pattern Description

An **architectural pattern** is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design patterns but have a broader scope. The architectural patterns address various issues in software engineering, such as computer hardware performance limitations, high availability and minimization of a business risk. Some architectural patterns have been implemented within software frameworks. I used a Client–server architecture and Observer Design Pattern in my application.A **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. Clients therefore initiate communication sessions with servers which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web. Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class *Observer* and a concrete class *Subject* that is extending class *Observer*.

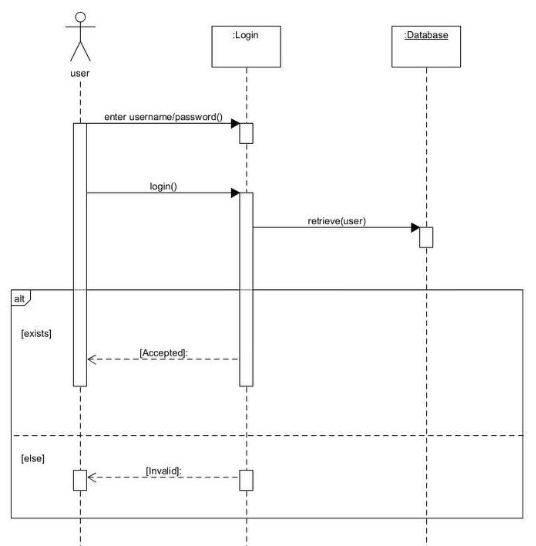
## 3.2 Diagrams

Package diagram:



# 4. UML Sequence Diagrams

Below is presented the sequence digram for log in action



# 5. Class Design

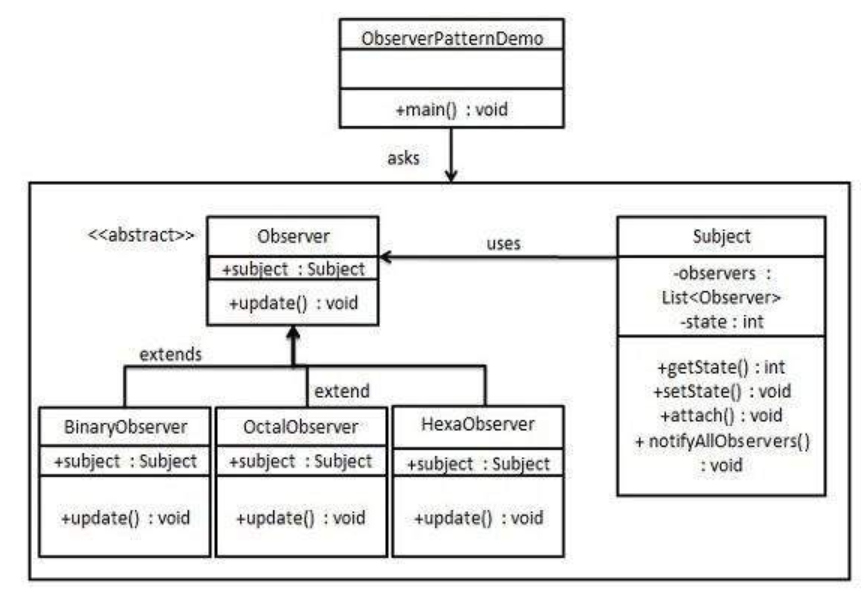
## 5.1 Design Patterns Description

Observer pattern:

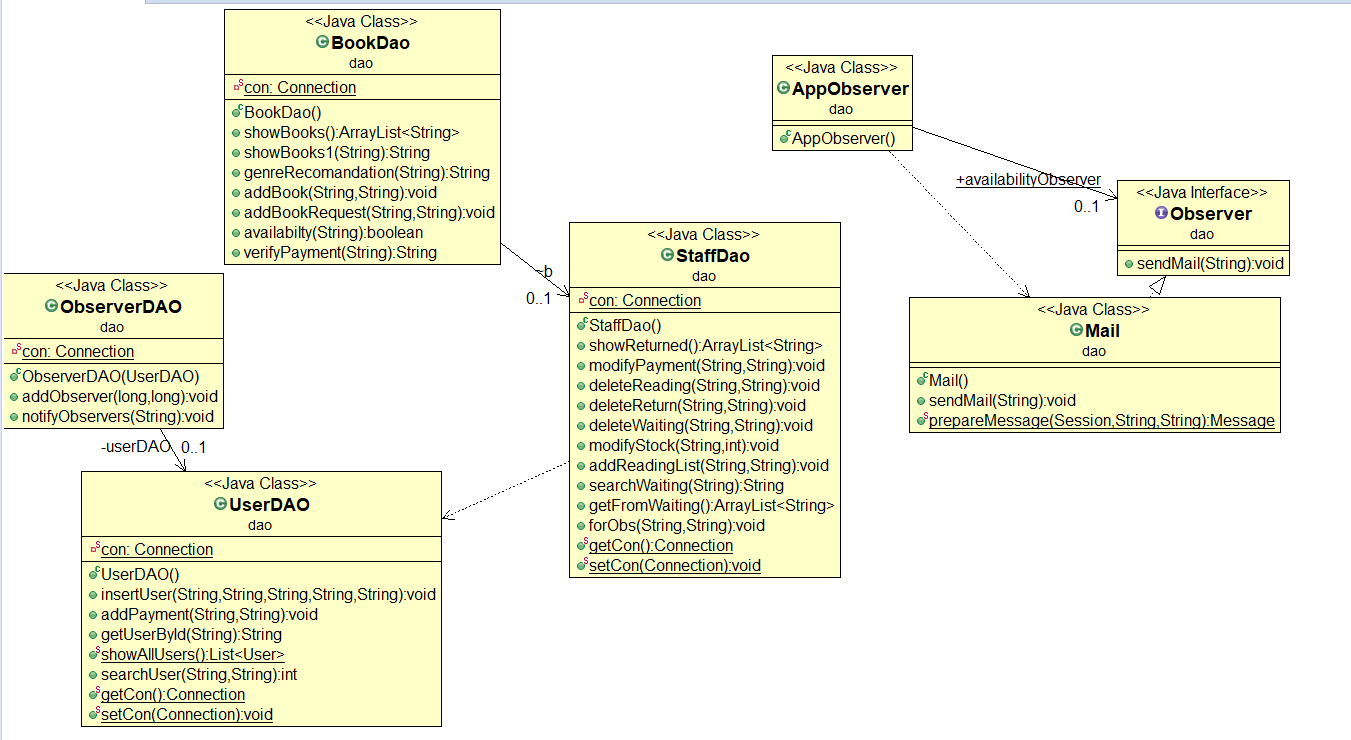
Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

Implementation example for Observer pattern:

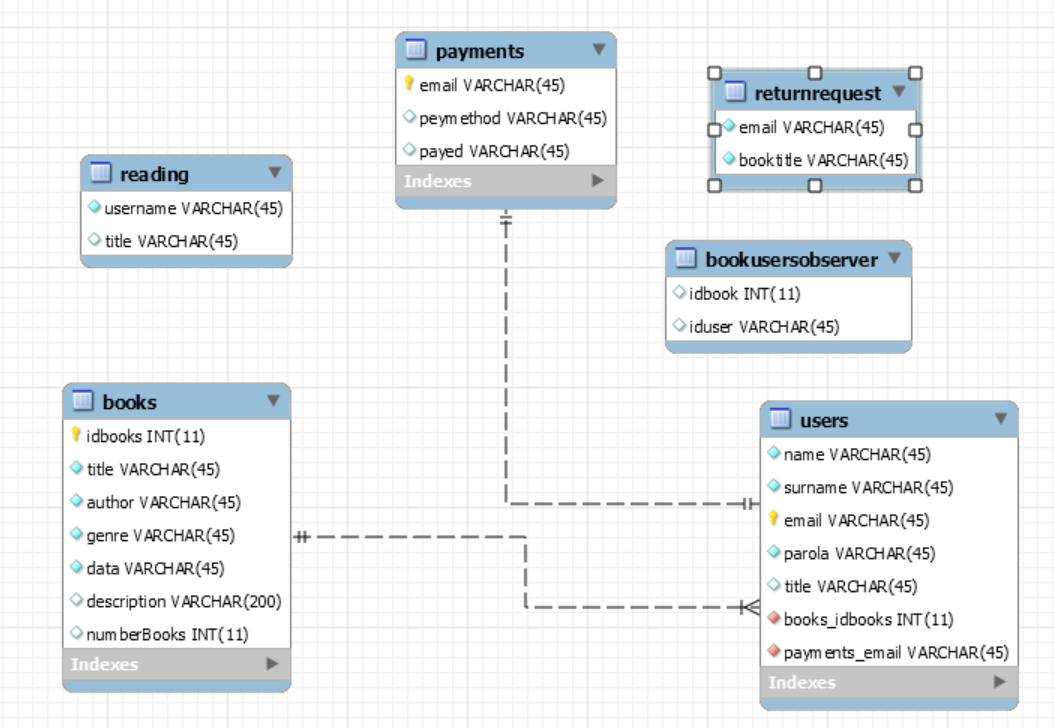
*ObserverPatternDemo*, our demo class, will use *Subject* and concrete class object to show observer pattern in action.



## 5.2 UML Class Diagram



# 6. Data Model



# 7. System Testing

*[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]*

# 8. Bibliography

* <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>
* <https://www.javatpoint.com/example-of-login-form-in-servlet>