Your Books Everywhere!

Analysis and Design Document

Student:Luca-Dan Adrian

**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

In order to implement a book management service I used JAVA objected oriented programming language. The app has two kinds of users: one is the client who can find, lend and return books, and the other one is the staff (or the librarian) who is able to manage the library.

A user is able to create an account, choose a payment plan and log in to search for books in the library. The client can also borrow books (if the books are available and if not he will join a waiting list).

The librarian or the staff can add, delete or update the books from the library and also can search for them by filtering. The books are filtered by the release date, author or title.

# Functional Requirements

The information about books, staff and users are saved in MySql database. The project has a model view controller architecture which is structured like this: Model, Data Access Object and Service and Controller and view.

The model has classes which include particularities and have the exact same fields as the tables from the database.

The data access object or DAO does the bond to the database, here we retrieve information based on some SQL statements.

The services executes actions which are sent to the controller layer. Services are executed with the help from data access object layer.

The controller send information to HTML files and here we are using operations like GET POST DELETE.

The view includes user interface HTML files which helps the users to interact with the library management application.

# Non-functional Requirements

2. Use-Case Model

*A picture containing text, map

Description generated with very high confidence*

Use case: Manage library

Level: User- goal level

Primary actor: Staff/Librarian

Main success scenario: The librarian manages to connect to the app using the log in page. Then he is able to manage the library by adding, removing or updating the stock of books and then he can filter the information from the database

Extension: If the librarian will not be able to connect, it will be impossible to access or to filter the information from the database

A close up of a logo

Description generated with very high confidence

Use case: Borrow a book and return it

Level: user-goal level

Primary actor: User/Client

Main success scenario: After the user manages to connect to the application using the log in page, he will be able to see the books which are available and not. The client will be able to see all the title of the books, including authors, genre and price. After then a user can borrow a books and after that to return them.

Extension: If a book is not available at a moment, the user must wait until it is and a message will be displayed.

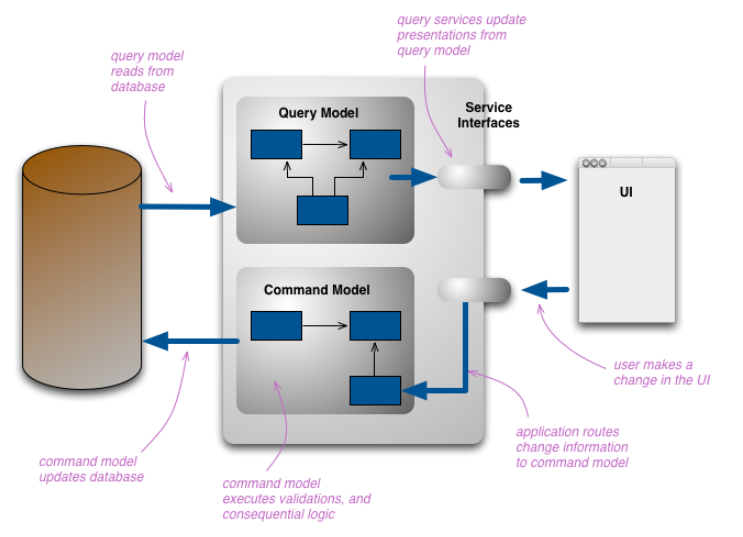
3. System Architectural Design

**3.1 Architectural Pattern Description**

CQRS stands for **Command Query Responsibility Segregation**.

When users interact with the information they use various presentations of this information, each of which is a different representation. Developers typically build their own conceptual model which they use to manipulate the core elements of the model.

The change that CQRS introduces is to split that conceptual model into separate models for update and display, which it refers to as Command and Query respectively following the vocabulary of Command Query Separation. The rationale is that for many problems, particularly in more complicated domains, having the same conceptual model for commands and queries leads to a more complex model that does neither well.

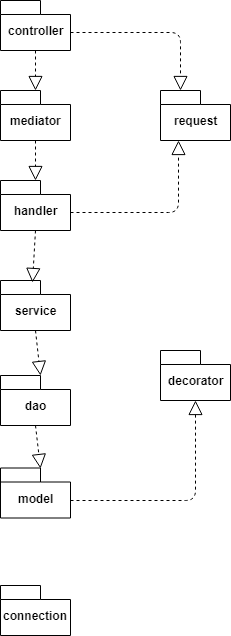


**3.2 Diagrams**

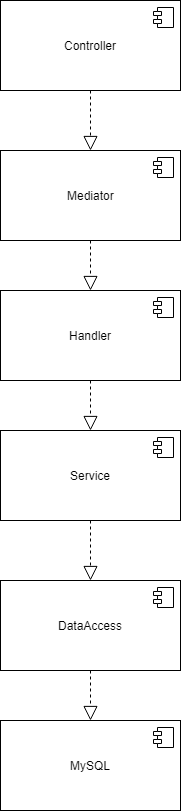
A screenshot of a social media post

Description generated with very high confidence

Package diagram

**

Component Diagram



4. UML Sequence Diagrams

5. Class Design

**5.1 Design Patterns Description**

**Mediator**

Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes

**Decorator**

Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

**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://martinfowler.com/bliki/CQRS.html>

<https://www.tutorialspoint.com/design_pattern/mediator_pattern.htm>

<https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm>