Book Management Application

Student: Casiana Stefana Campean

**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 4

3. System Architectural Design 5

4. UML Sequence Diagrams 8

5. Class Design 8

6. Data Model 9

7. System Testing 10

8. Bibliography 10

1. Requirements Analysis

# Assignment Specification

For this assignment we need to build a book management service.

A user should be able to create an account, choose a payment plan and log in 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 it’s 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 assign the book to the next user in the waiting list after validation by the library staff.

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

# Functional Requirements

The functional requirements are:

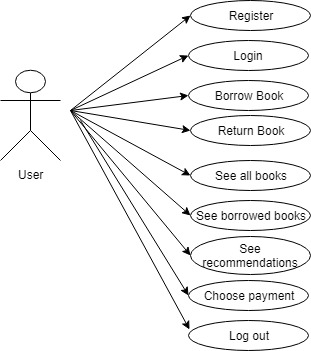
* to implement and test the application
* to use layered architecture
* use a factory method for building user recommendations by trends, genres or topics
* Store data in a database
* Validate inputs before submitting he data and saving it in the database

# Non-functional Requirements

The functional requirements are:

* To commit the work to my Git repository and to do it iteratively as I progress
* To use OOP language
* Use layered architecture

2. Use-Case Model



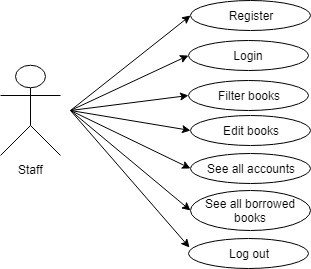
Use case: Borrow Book

Level: User-goal level

Primary actor: User

Main success scenario: If he doesn’t have an account, he register and he choose a payment plan. Then he login, he selects See all books button, he selects Borrow button, he introduces the correct book’s id, then a book is added to his borrowed books list. He can see his new borrowed book if he presses the Back button, he presses the show My Books button, then he sees all his borrowed books and also there he can return any book.

Extensions: He presses the Borrow button, he introduces an incorrect book id, then he gets a message saying the book id is invalid.



Use case: Filter books

Level: Staff-goal level

Primary actor: Staff

Main success scenario: If he doesn’t have an account, he choose to register. Then he enter his account, he presses filter by button, he introduce the correct text in the title, author, date or genre text field and then he presses the filter button by title, author, date or genre and then he can see that data.

Extensions: He presses the Filter by button and he doesn’t introduce the existent values in the text fields, so he gets a message that the data is incorrect.

3. System Architectural Design

**3.1 Architectural Pattern Description**

The architectural pattern required for this project is Layered Architecture Pattern. This project has four layers:

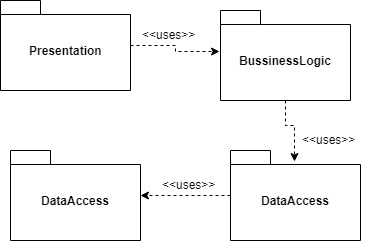
* Presentation Layer(Presentation): this layer has the user interface implemented; it uses the business layer for all operations
* Business Layer(Business Logic): this layer uses the dataAccess layer to manipulate data in the database, to provide operations for the presentation layer.
* Persistence Layer(dataAccess): this layer uses the data access layer to make CRUD operations on the database.
* Database Layer(models): this layer has classes representing the tables in the database.



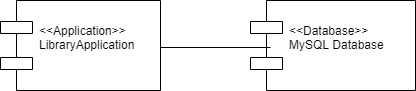
Figure 1 illustrates the diagram for the layered architecture.

**3.2 Diagrams**

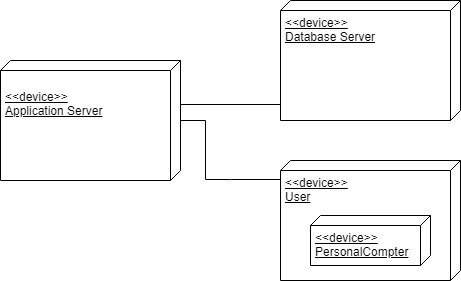
***Package diagram***

******

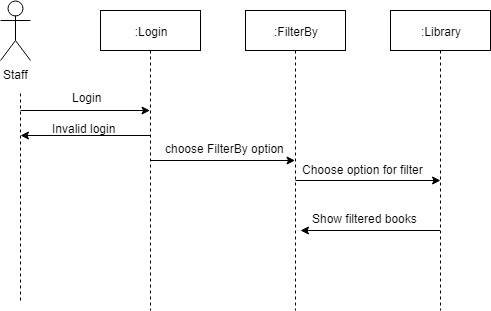
***Component diagram***

******

***Deployment diagram***

******

4. UML Sequence Diagrams



This sequence diagram represents a case in the application. The user of this application is the staff user, who can filter the books by title, author, date or genre.

5. Class Design

**5.1 Design Patterns Description**

The design pattern required for this project is Factory Design Pattern, a creation pattern.

Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

**Implementation**

**Step 1**

Create a Factory Interface

public interface Recommendation {

ArrayList<Book> getReccomendation(String s);

}

**Step 2**

Create concrete classes implementing the same interface:

public class RecommendationByPopularity implements Recommendation

public class RecommendationByGenre implements Recommendation

**Step 3**

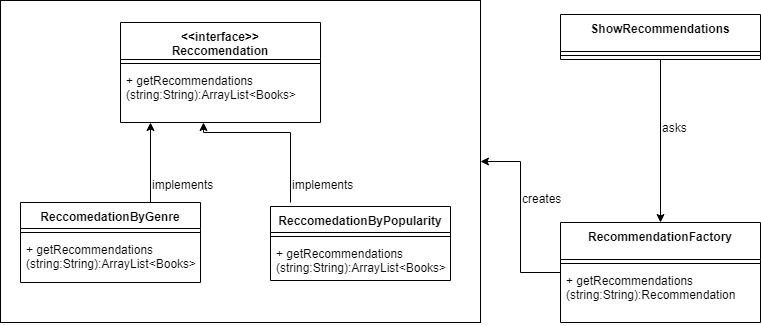
Create a Factory to generate object of concrete class based on given information:

RecommendationFactory, where I have a Factory method.

**Step 4**

Create objects, this part is in the presentation layer, in ShowRecommendations.

**5.2 UML Class Diagram**



This represents the UML class diagram for the Factory Design pattern implemented in this project. The implementation is described above.

6. Data Model

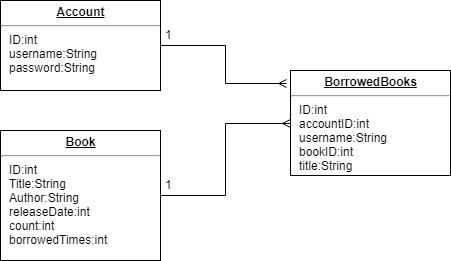
The data model is represented as Java classes and as database tables.

In this project, the data model is represented by these classes in Java/tables in DB:

-Account

-Book

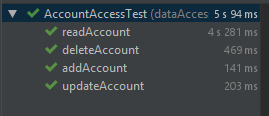
-BorrowedBooks



This diagram represents the relation between tables. The borrowedBooks table contains all the details about the existent borrowed books in the library application. Between book and borrowedBooks and account and borrowedAccount is a one to many relationship, because more than one book or account can appear in the borrowedBooks table.

7. System Testing

In order to test the project, I implemented Junit test for the class AccountAccess, testing insert, update, read and delete methods.

**

I considered it was enough to test this class because the other access classes are written the same.

The other methods in other classes were tested in main.

8. Bibliography

[1] Hibernate Tutorial: <https://www.youtube.com/playlist?list=PLEAQNNR8IlB7fNkRsUgzrR346i-UqE5CG>

[2] Factory Method:

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

[3] Layered Architecture:

<https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html>

[4] Reflection

<http://tutorials.jenkov.com/java-reflection/index.html>