Book store

Analysis and Design Document

Student: Sergiu Coca

**Group: 30233**

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

Use Java/C# API to design and implement an application for the employees of a book store. The application should have two types of users (a regular user represented by the book store

employee and an administrator user) which have to provide a username and a password in order

to use the application.

# Functional Requirements

The regular user can perform the following operations:

- Search books by genre, title, author.

- Sell books.

The administrator can perform the following operations:

- CRUD on books (book information: title, author, genre, quantity, and price).

- CRUD on regular users’ information.

- Generate two types of reports files, one in pdf format and one in csv format, with the

books out of stock.

# Non-functional Requirements

The information about users, books and selling will be stored in multiple XML files. Use

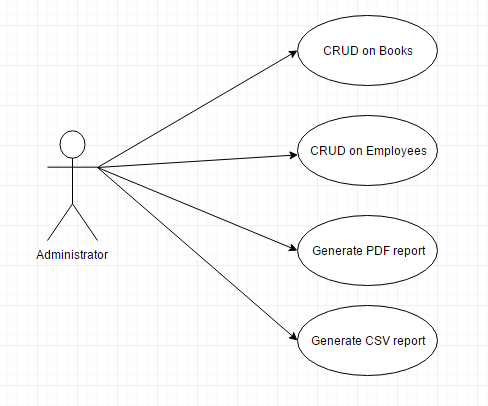
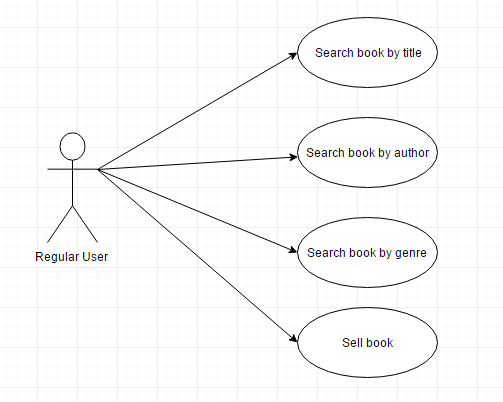
the Model View Controller in designing the application. Use the Factory Method design

pattern for generating the reports.

All the inputs of the application will be validated against invalid data before submitting

the data and saving it.

2. Use-Case Model

*Use case:* Sell book

*Level:* Sub-function

*Primary actor:* Employee

*Main success scenario:* The requested book is in stock and the customer has money to buy it.

*Extensions:* It is possible that the requested book is not available.

3. System Architectural Design

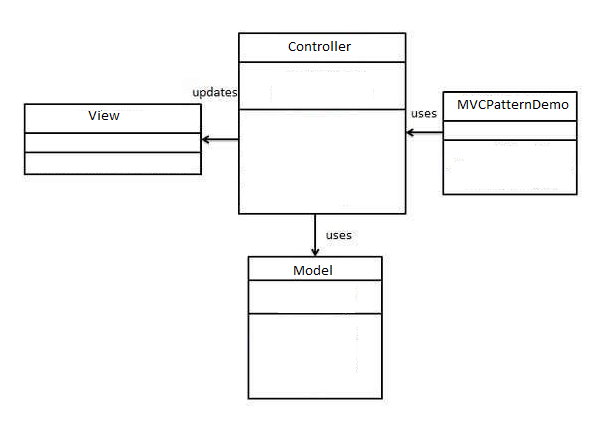
**3.1 Architectural Pattern Description**

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

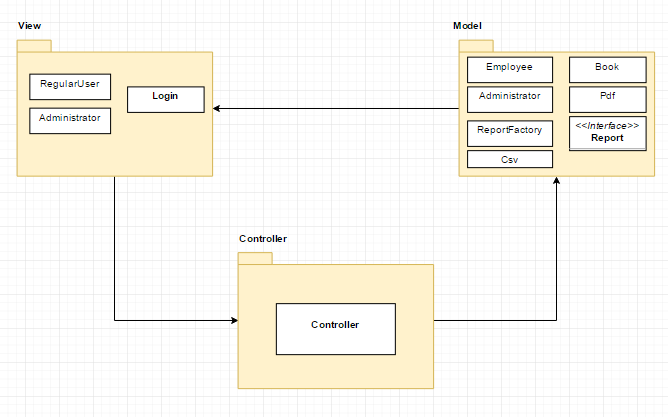
* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

**3.2 Diagrams**

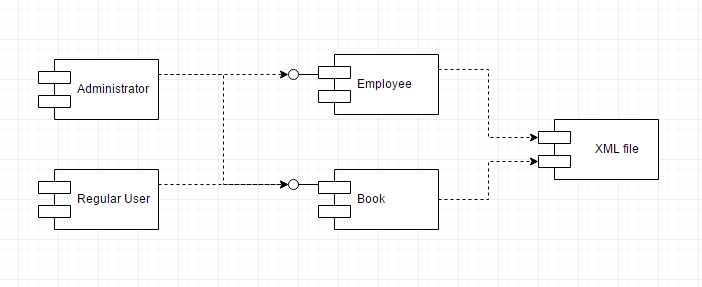
**MVC diagram**

****

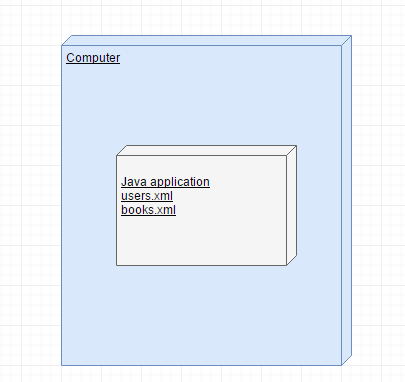
**Package diagram**

****

**Component diagram**

****

**Deployment diagram**

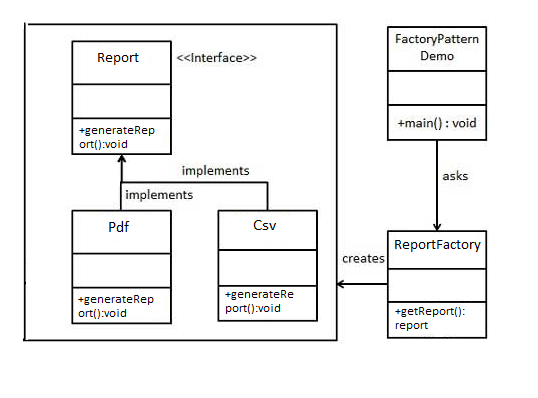
****

4. UML Sequence Diagrams

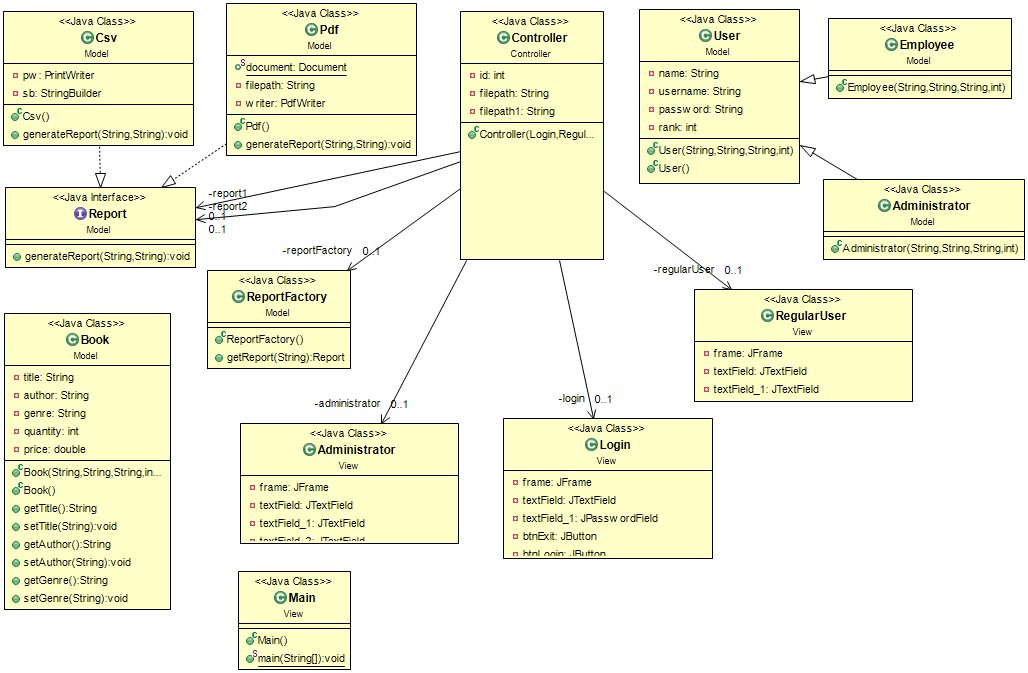
5. Class Design

**5.1 Design Patterns Description**

For generating a report I used Factory pattern. 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. I’ve created Report interface and Pdf,Csv classes implementing the Report interface. After that, I’ve created a factory class ReportFactory. The Controller will use ReportFactory to get a Report object(PDF/CSV).



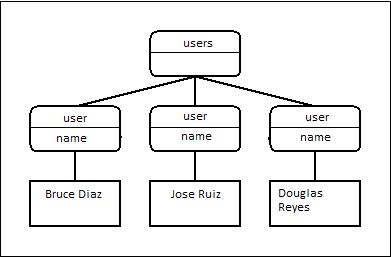
**5.2 UML Class Diagram**



6. Data Model

The data model for XML is very simple - or very abstract, depending on one's point of view. XML provides no more than a baseline on which more complex models can be built. All those more restricted applications will share some common invariants, however, and it is those that are given below.

Think of an XML document as a linearization of a tree structure. At every node in the tree there are several character strings. The tree structure and the character strings together form the information content of an XML document. Almost everything will follow naturally from that. Some of the characters in the document are only there to support the linearization, others are part of the information content.



7. System Testing

For CRUD operations I checked if the data exists in XML files. For example, if you want to search a book, the application verifies if the book exists. If a condition are not satisfied, the applications shows a message like “The book was not found!”.

8. Bibliography

<http://www.mkyong.com/tutorials/java-xml-tutorials/>

<https://www.tutorialspoint.com/index.htm>

<https://stackoverflow.com/>