Book Store

Student: Abrudan Darius

**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 4

3. System Architectural Design 5

4. UML Sequence Diagrams 7

5. Class Design 7

6. Data Model 8

7. System Testing 9

8. Bibliography 10

1. Requirements Analysis

# Assignment Specification

Design and implement an application for the employees of a book store using Java/C# API. 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

Some of the non-functional requirements that are present in the application are privacy and security while using a log in and a log out mode for the application, so that not everybody could use this application, just the employees of the book store and the administrator who have control on the employees.

Efficiency and usability are another non-functional requirement that are present by the functionality of getting a specific type of report (pdf format and csv format) with the books out of stock. Another types of reports could be easily added because of the facility of the factory pattern which is used in order to manage multiple types of reports files.

Maintainability and extensibility can be also seen by having the possibility of adding new books in the store if there are any new books that are published so the store will be updated in order to keep up with the progress. In time, some employees can resigned and others people can take their places, functionality maintained by the administrator, which has the ability to hire and fire who he wants.

2. Use-Case Model

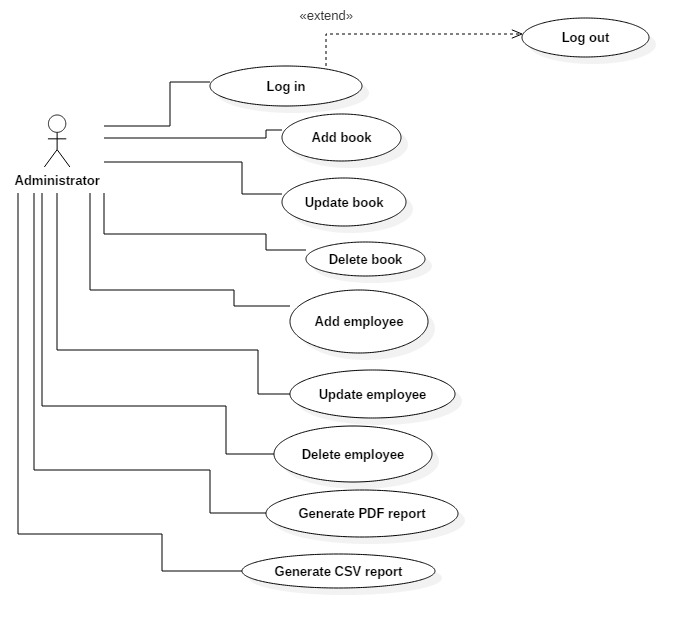
Use case: sell book

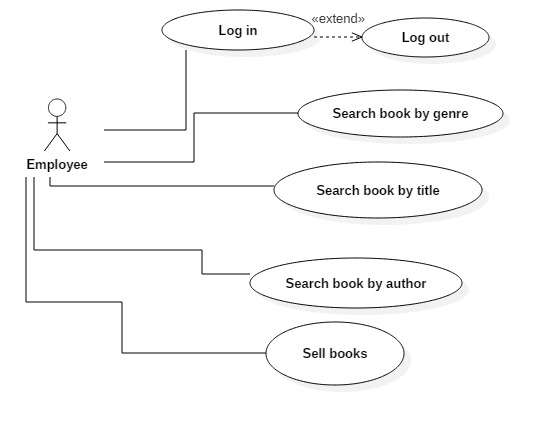
Level: user-goal level

Primary actor: employee

Main success scenario: there are enough books in the store to sell them to a particular client

Extensions: not enough books in the store





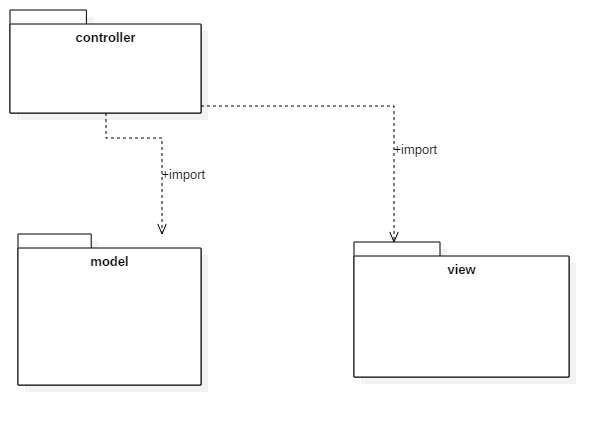
3. System Architectural Design

**3.1 Architectural Pattern Description**

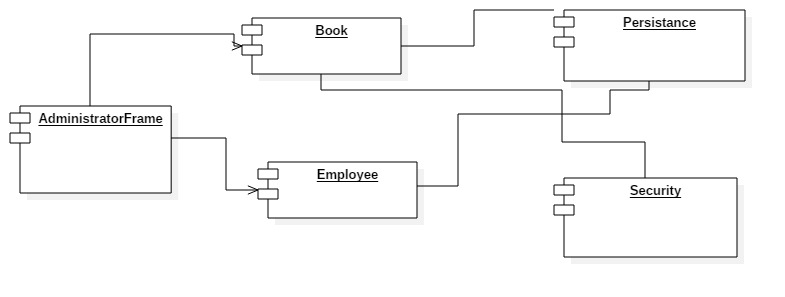
I used the Model View Controller Architectural Pattern or so called MVC Pattern. Model represents objects which contain data. View represents the visualization of the data that model contains. View contains the classes corresponding to the frames of the GUI. Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It also keeps view and model separate.

**3.2 Diagrams**

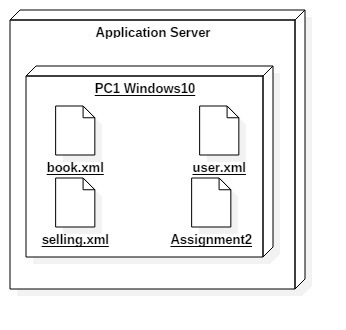
In the diagrams below, it is shown how the model view controller architectural pattern is applied to the book store application. Each component is represented by a package with the same name.

****

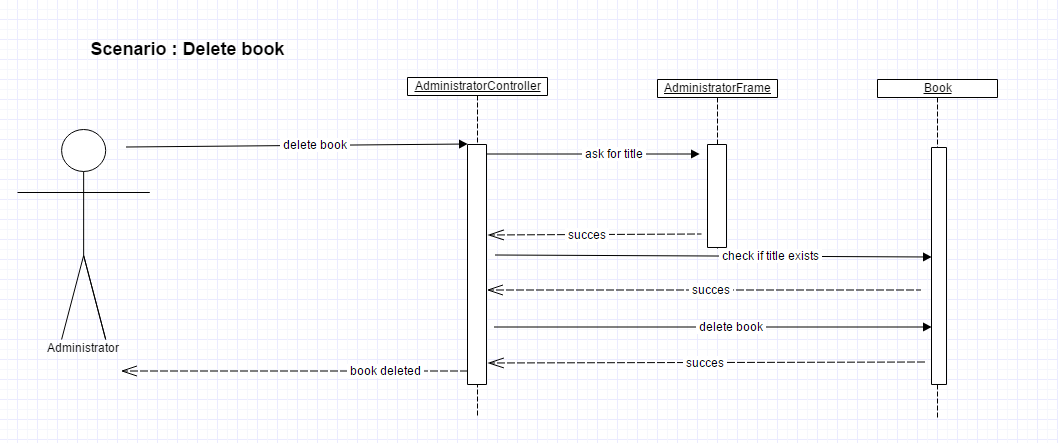
Component diagram:



Deployment diagram:

****

4. UML Sequence Diagrams

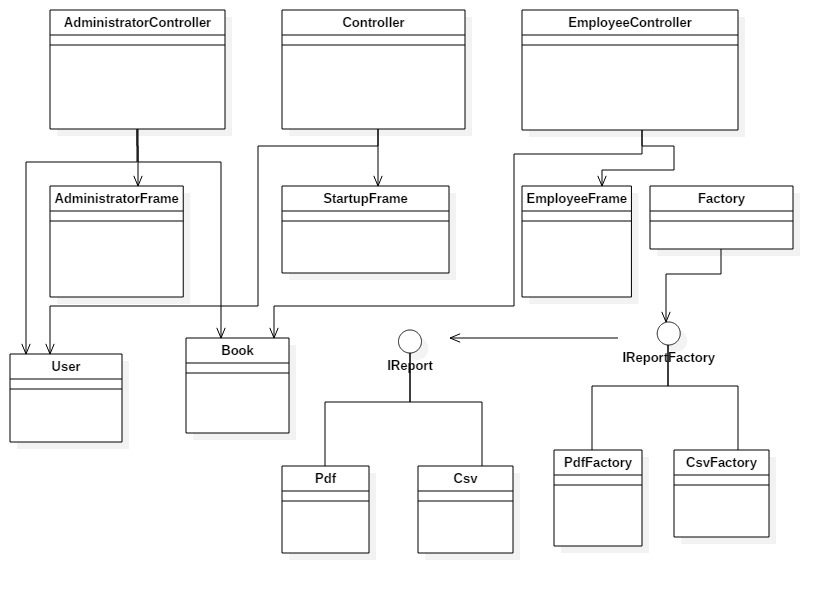


5. Class Design

**5.1 Design Patterns Description**

In the book store application created, I used the Factory Pattern for the reports of the books which are out of stock. It is one of the most used creational patterns because the objects are created without exposing the logic to the client and refer to newly created object using a common interface (IReport). Pdf and Csv classes implement the IReport interface and have a generate method for writing the specific report of books out of stock.

**5.2 UML Class Diagram**

**

6. Data Model

The data are stored in 3 xml files, one for books, one for users and one for selling information. The book xml file contains for each book an id, title, author, genre, quantity and also the price for that book. The user xml file has for each user an id, username, password and type, which can be administrator or employee. Selling xml file contains data about the books that where sold: book id, title, author, genre, and number of copies, price and the total price that needs to be paid.

book.xml:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<store>

<book id="1">

<title>Ion</title>

<author>Liviu Rebreanu</author>

<genre>epic</genre>

<quantity>55</quantity>

<price>30</price>

</book>

</store>

user.xml:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<store>

<user id="1">

<username>Sorin</username>

<password>seful</password>

<type>administrator</type>

</user>

</store>

selling.xml:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<selling>

<book id="1">

<title>Ion</title>

<author>Liviu Rebreanu</author>

<genre>epic</genre>

<copy\_numbers>10</copy\_numbers>

<price>30</price>

<total\_price>300</total\_price>

</book>

</selling>

7. System Testing

For the correct functionality of the application, I designed a couple of tests such that all the inputs of the application (inputs introduced in J Text Fields) will be validated against invalid data before submitting the data and saving it in the xml files. If a J Text Field is not completed, or if there is required a number and the user will introduce a string, it will output a J Option Pane with an appropriate message so that the user could know exactly which field was not completed correct. I also verified for the log in Startup Frame such that the username exist and the password for a specific username is correct. In case of such an inconsistence, a message will appear.

8. Bibliography

1 .<https://stackoverflow.com/>

2. <https://docs.oracle.com/javase/tutorial/>

3. <https://www.tutorialspoint.com/>