Bookstore Application
Analysis and Design Document

Student: Dregan Anda-Denisa
Group: 30233

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

The assignment specification looks like this:

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.

The regular user can perform the following operations:

- Search books by genre, title, author.
- Sell books.

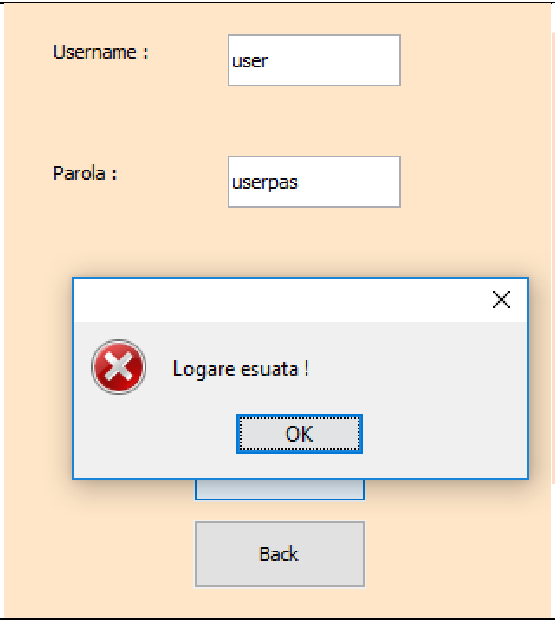
The administrator can perform the following operations:

- 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.

1.2 Functional Requirements

Firstly, the two users, user and administrator, must enter a valid name and password in order to make the operations allowed for each of them. If logging was not made, an error message appears and they can still try to log. Otherwise, there is a new window, where they can operate.

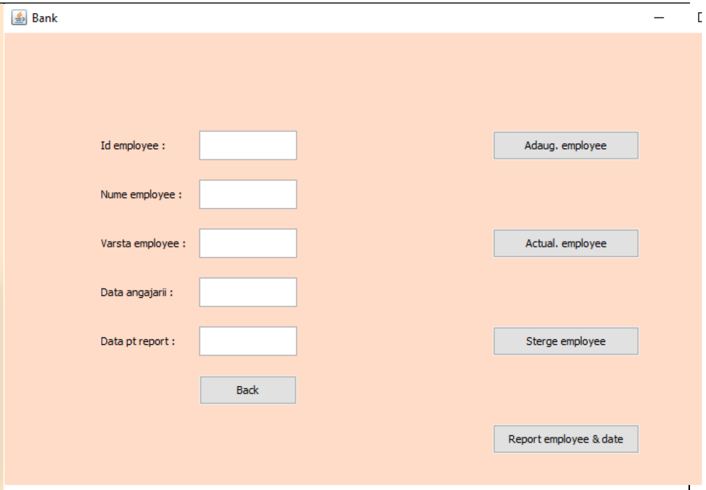
Below, you can see two cases: failed login admin, admin logged correctly.



Username : user

Parola : userpas

Logare esuata !



Id employee :

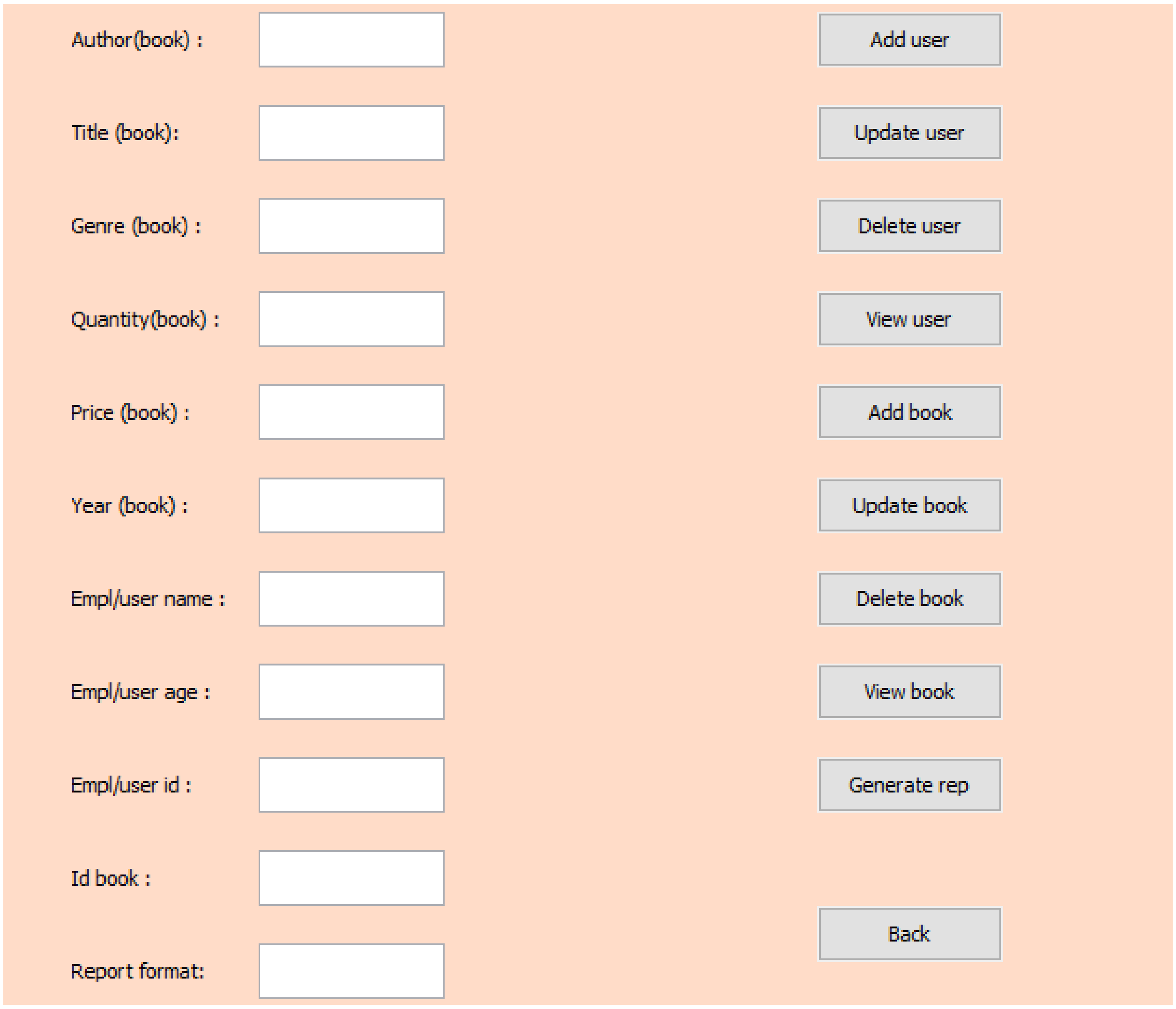
Nume employee :

Varsta employee :

Data angajarii :

Data pt report :

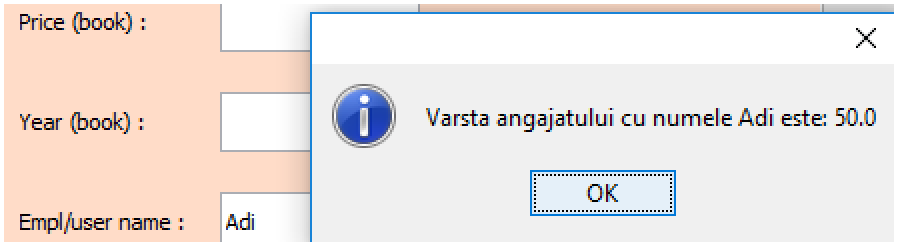
For example, if I logged in as admin, I can execute CRUD operations for employees, books and generate reports with those out of stock, as seen in the following picture:



When I am logged successfully as a user, I have several possibilities: searching books by title,genre and author, accompanied by information or alert messages. In addition, I can sell a book, by typing the title and the quantity the client wants.

Both users have access to the data containing details about books and employees (the last ones are available only for the admin user), just clicking the buttons “View book “ and ”View user” . and typing the id.

See the following example for employees’(users’) case:



Varsta angajatului cu numele Adi este: 50.0

1.3 Non-functional Requirements

While the functional requierements describe what the system should do, the non-functional ones describe how the system works.

Among the known nonfunctional requierements, this project follows

*the minimum respons time : total amount of time it takes to respond to a request for service is small, meaning that if users make changes to the database tables available in the application interface will be updated in real time to verify the correctness of operations at the same time; it led to the next requirement :

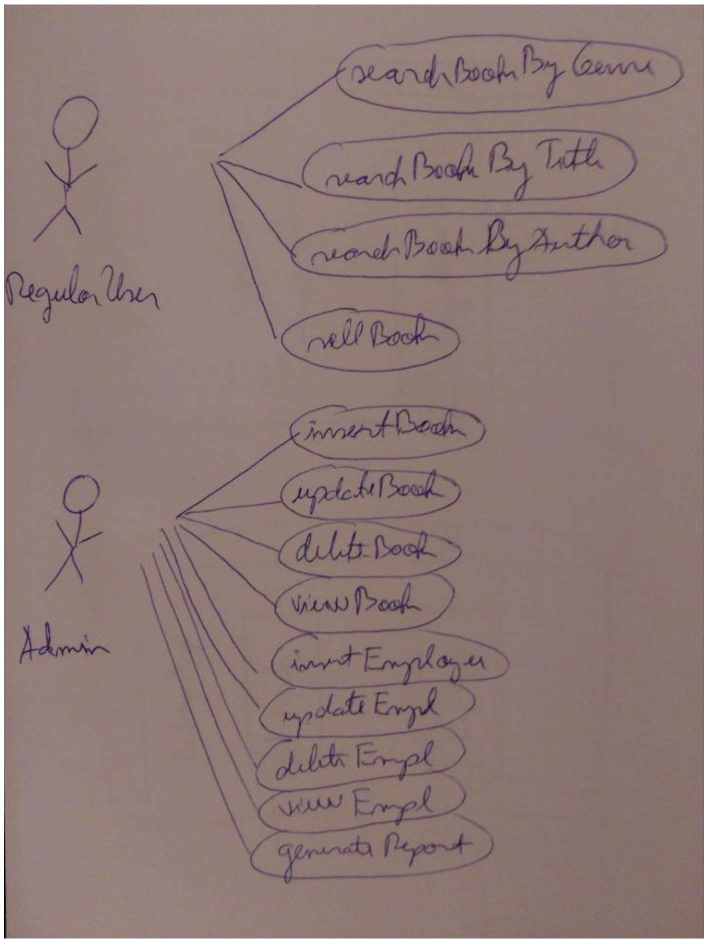
*scalability: this system is able to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth

*security : to avoid situations where some enivous users change data that should not change, everyone must log into their account

*usability : it is easy to operate with this system, because the interface is friendly and concise.

2. Use-Case Model

The use case diagrams are showed in the picture below:



Use case : sell a book
Level: subfunction
Primary actor : only regular user can do this activity of selling books
Main success scenario : the user log correctly into application, introduces all the infos required for selling a book : the title and the quantity. Then click on the button “Sell book” and the modification is done automatically. By clicking on the button “Search book by title”, the user see the
Extensions: a first failure in this use case could be user not knowing the username and the password for his account on the application.

3. System Architectural Design

3.1 Architectural Pattern Description

I implemented the solution using Model-View-Controller architectural pattern.
This pattern proposes three main components or objects to be used in software development:
A Model , which represents the underlying, logical structure of data in a software

application and the high-level class associated with it. This object model does not contain any information about the user interface.

In this stage, the application contains a class named Model, where there are several methods like “getBookTitleByAuthor”,”getBookGenreByTitle”, “getEmployeeAgeByName”, which return some useful details, using xpath, from the xml files.

A View , which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth).

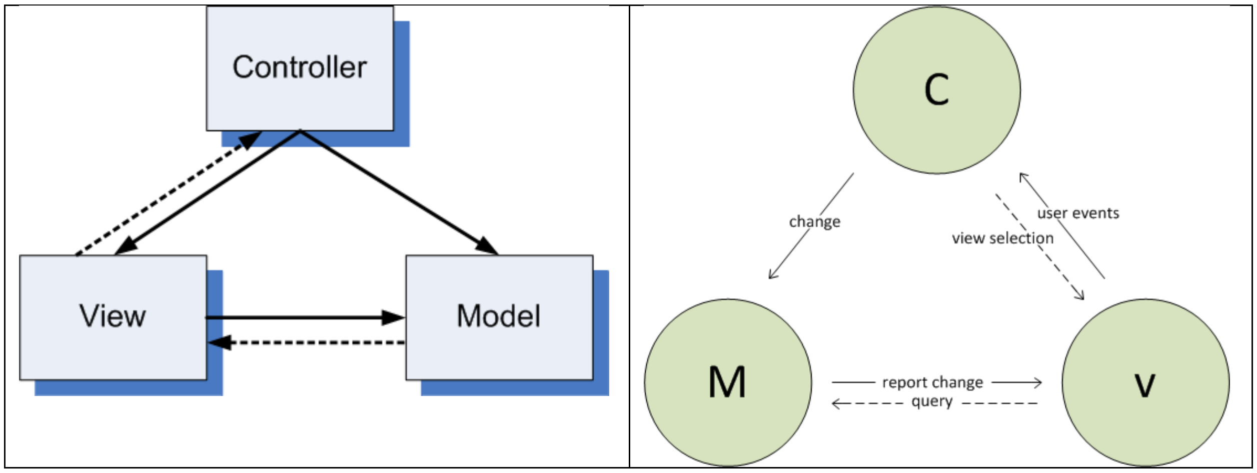
In this stage, the application has a class named GUI. In gui constructor, the buttons, textfields and the labels are arranged on the screen and the other methods return the values introduces in the textfields by the user.

A Controller , which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

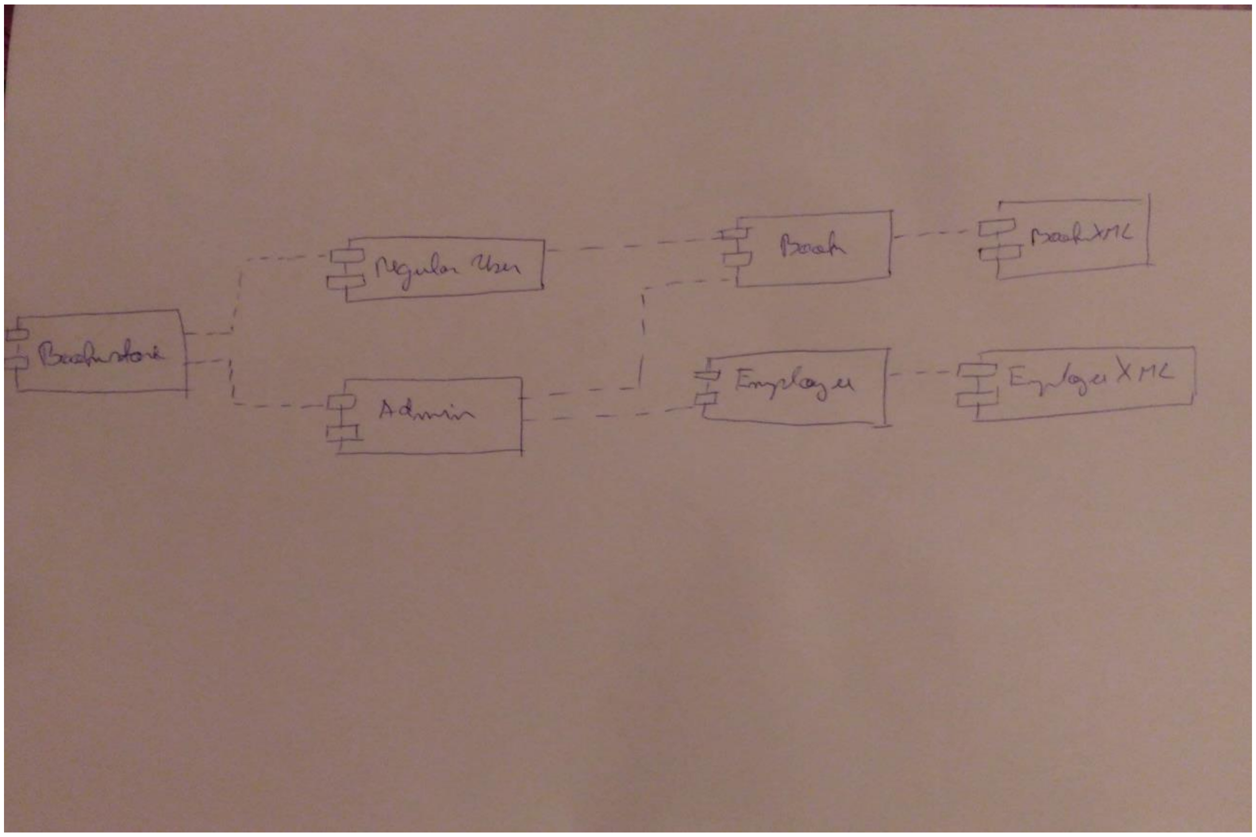
This project has a class named Controller, which calls methods from both view and model classes and processes the inputs. For example, the viewBook method takes from gui the title introduces by the user and from the model, the author, genre, price and the quantity for that book and prints all these in a JOptionPane.

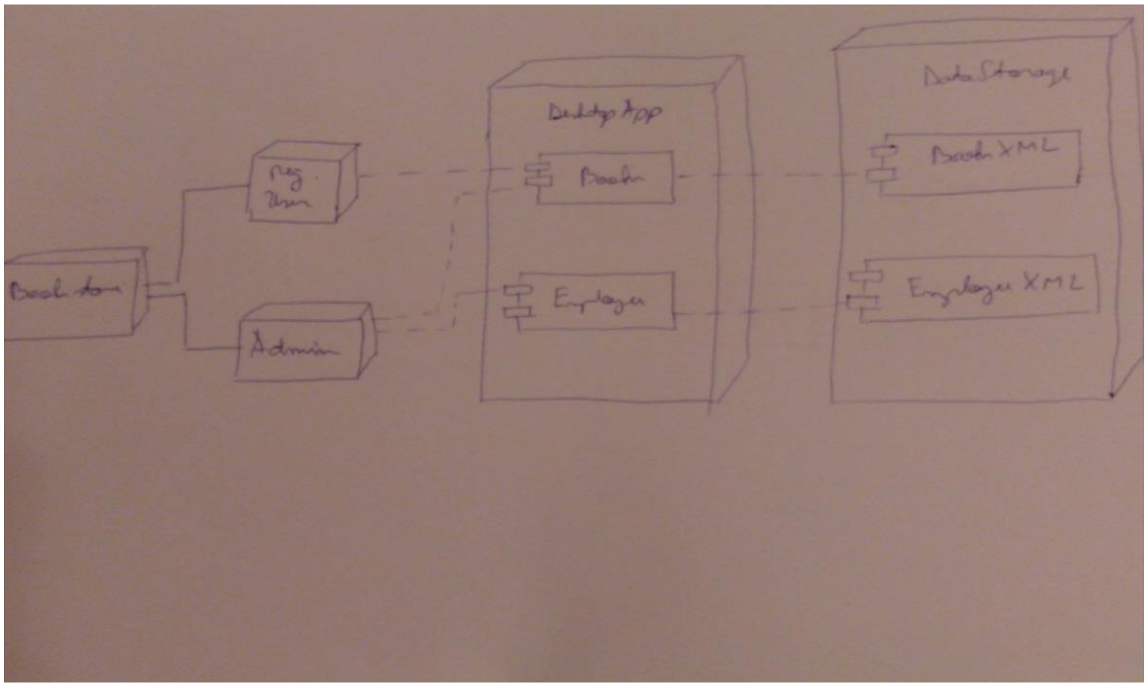
3.2 Diagrams

There are some schemas that show how the above architectural pattern are applied on this system :



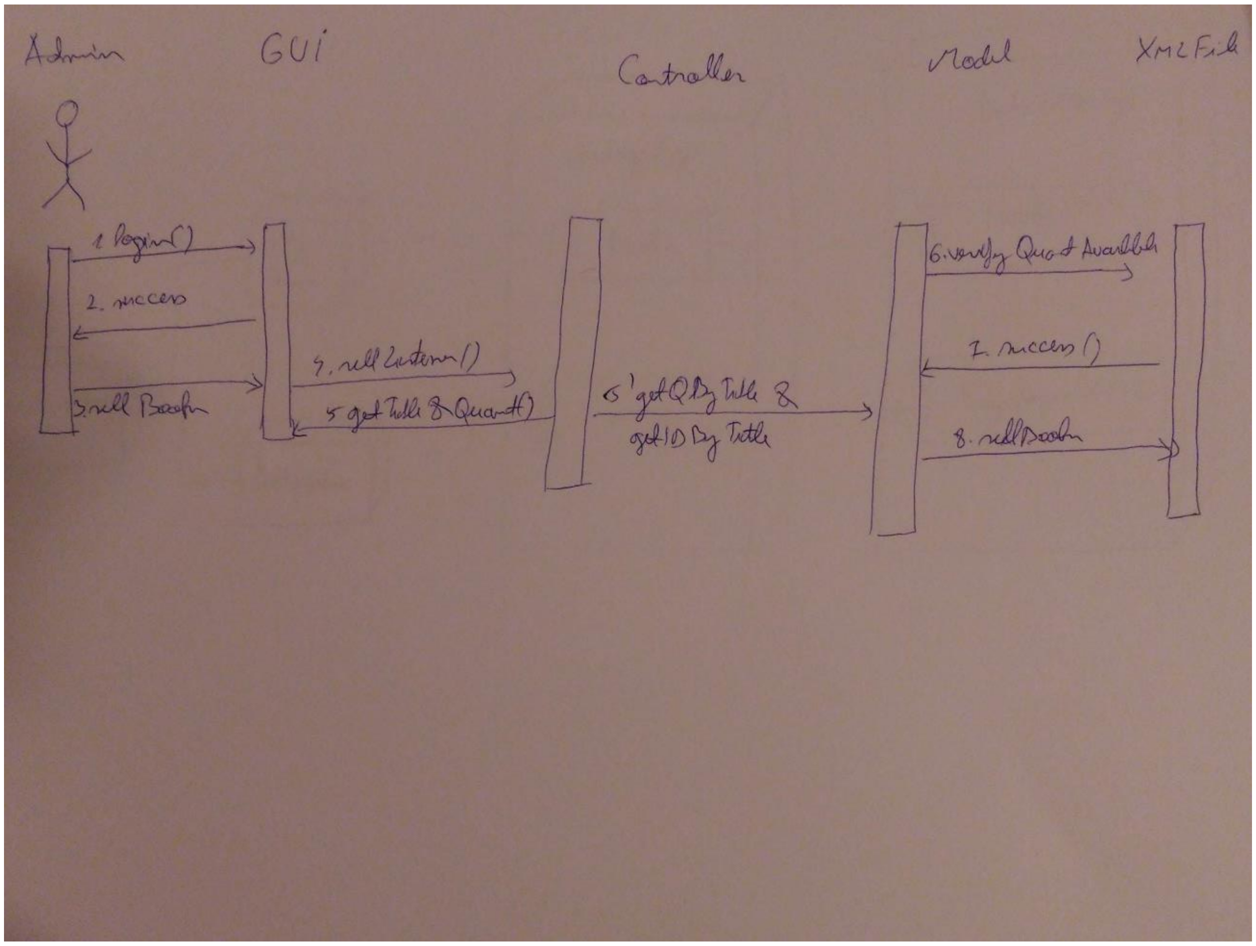
The pictures below show the component and the deployment diagrams for the application:





4. UML Sequence Diagrams

The following sequence diagram represents the scenario of selling a book. The regular user successfully log into his application account, introduces the title and the quantity, and by clicking on “sell book”, the operation is made if the quantity is enough.



5. Class Design

5.1 Design Patterns Description

As design pattern I implemented in this application the Factory Method pattern, for generating reports with books out of stock.
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.

I created an interface named Report, with one method : generate. I then created the concrete classes Report1 and Report2 that implements the interface. The ReportFactory concrete class contains the following lines of code:

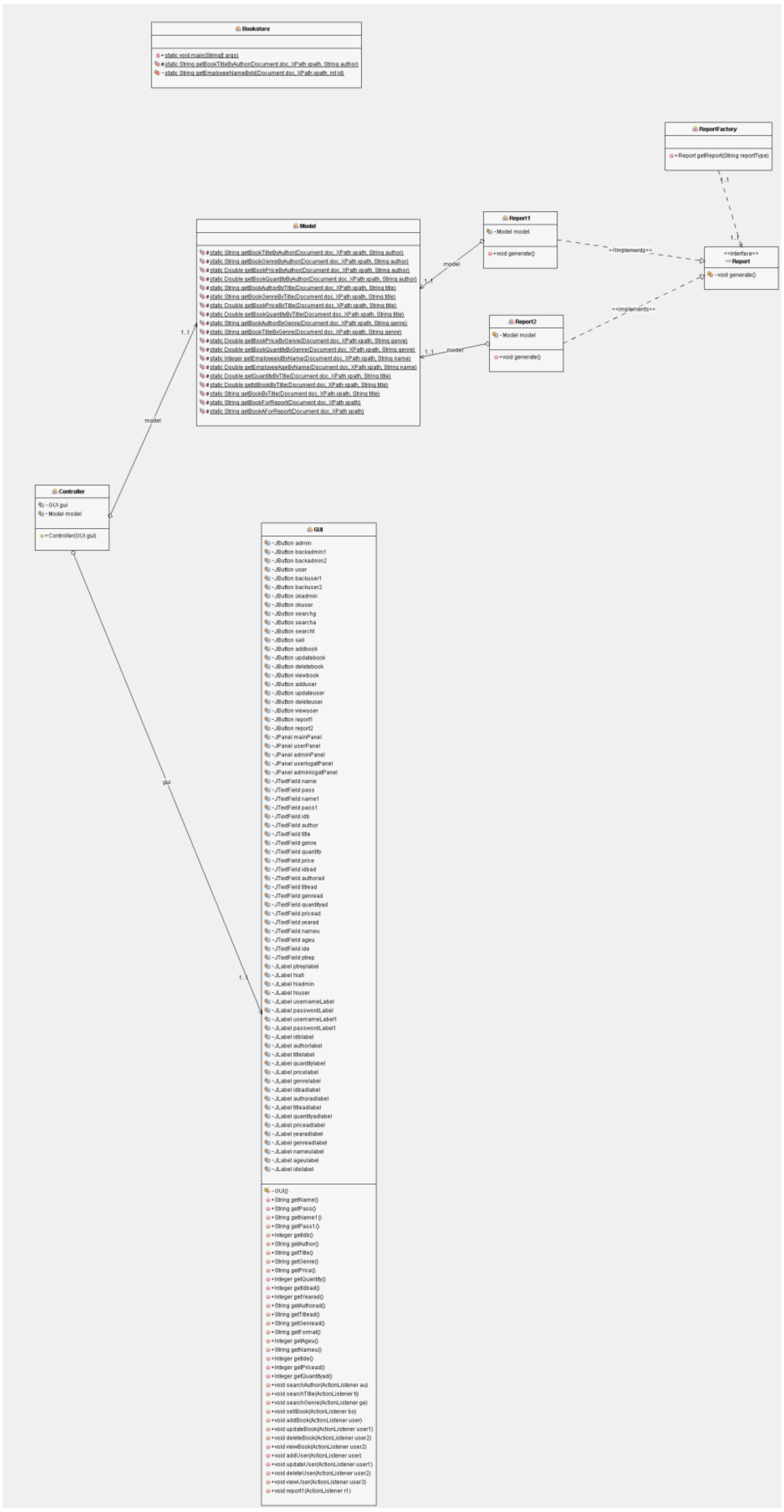
```
public Report getReport(String reportType){
    if(reportType == null){
      return null;
    }
    if(reportType.equalsIgnoreCase("pdf")){
      return new Report1();

    } else if(reportType.equalsIgnoreCase("csv")){
      return new Report2();
}
     return null;
  }
```
.

Finally, in class Controller it is a method that calls the methods from Report1 or Report2, depending on the desired file format.

5.2 UML Class Diagram

The following picture shows the class diagram of the application. It is easy to understand the concepts approached until now, because the diagram is structured as in the patterns' schema and are visible the layers and their incorporated classes.


Bookstore
+ static void main(String[] args)
# static String getBookTitleByAuthor(Document doc, XPath xpath, String author)
- static String getEmployeeNameById(Document doc, XPath xpath, int id)
ReportFactory
+ Report getReport(String reportType)
1..1
1..1
Model
# static String getBookTitleByAuthor(Document doc, XPath xpath, String author)
# static String getBookGenreByAuthor(Document doc, XPath xpath, String author)
# static Double getBookPriceByAuthor(Document doc, XPath xpath, String author)
# static Double getBookQuantityByAuthor(Document doc, XPath xpath, String author)
# static String getBookAuthorByTitle(Document doc, XPath xpath, String title)
# static String getBookGenreByTitle(Document doc, XPath xpath, String title)
# static Double getBookPriceByTitle(Document doc, XPath xpath, String title)
# static Double getBookQuantityByTitle(Document doc, XPath xpath, String title)
# static String getBookAuthorByGenre(Document doc, XPath xpath, String genre)
# static String getBookTitleByGenre(Document doc, XPath xpath, String genre)
# static Double getBookPriceByGenre(Document doc, XPath xpath, String genre)
# static Double getBookQuantityByGenre(Document doc, XPath xpath, String genre)
# static Integer getEmployeeIdByName(Document doc, XPath xpath, String name)
# static Double getEmployeeAgeByName(Document doc, XPath xpath, String name)
# static Double getQuantityByTitle(Document doc, XPath xpath, String title)
# static Double getIdBookByTitle(Document doc, XPath xpath, String title)
# static String getBookByTitle(Document doc, XPath xpath, String title)
# static String getBookForReport(Document doc, XPath xpath)
# static String getBookAForReport(Document doc, XPath xpath)
Report1
- Model model
+void generate()
model
1..1
<<implements>>
<<interface>>
Report
-void generate()
Report2
- Model model
+void generate()
model
1..1
<<implements>>
1..1
model
Controller
- GUI gui
- Model model
+ Controller(GUI gui)
gui
1..1
GUI
-JButton admin
-JButton backadmin1
-JButton backadmin2
-JButton user
-JButton backuser1
-JButton backuser2
-JButton okadmin
-JButton okuser
-JButton searchg
-JButton searcha
-JButton searcht
-JButton sell
-JButton addbook
-JButton updatebook
-JButton deletebook
-JButton viewbook
-JButton adduser
-JButton updateuser
-JButton deleteuser
-JButton viewuser
-JButton report1
-JButton report2
-JPanel mainPanel
-JPanel userPanel
-JPanel adminPanel
-JPanel userlogatPanel
-JPanel adminlogatPanel
-JTextField name
-JTextField pass
-JTextField name1
-JTextField pass1
-JTextField idb
-JTextField author
-JTextField title
-JTextField genre
-JTextField quantity
-JTextField price
-JTextField idbad
-JTextField authorad
-JTextField titlead
-JTextField genread
-JTextField quantityad
-JTextField pricead
-JTextField yearad
-JTextField nameu
-JTextField ageu
-JTextField ide
-JTextField ptrep
-JLabel ptreplabel
-JLabel hiall
-JLabel hiadmin
-JLabel hiuser
-JLabel usernameLabel
-JLabel passwordLabel
-JLabel usernameLabel1
-JLabel passwordLabel1
-JLabel idblabel
-JLabel authorlabel
-JLabel titlelabel
-JLabel quantitylabel
-JLabel pricelabel
-JLabel genrelabel
-JLabel idbadlabel
-JLabel authoradlabel
-JLabel titleadlabel
-JLabel quantityadlabel
-JLabel priceadlabel
-JLabel yearadlabel
-JLabel genreadlabel
-JLabel nameulabel
-JLabel ageulabel
-JLabel idelabel
-GUI()
+String getName()
+String getPass()
+String getName1()
+String getPass1()
+Integer getIdb()
+String getAuthor()
+String getTitle()
+String getGenre()
+String getPrice()
+Integer getQuantity()
+Integer getIdbad()
+Integer getYearad()
+String getAuthorad()
+String getTitlead()
+String getGenread()
+String getFormat()
+Integer getAgeu()
+String getNameu()
+Integer getIde()
+Integer getPricead()
+Integer getQuantityad()
+void searchAuthor(ActionListener au)
+void searchTitle(ActionListener ti)
+void searchGenre(ActionListener ge)
+void sellBook(ActionListener bo)
+void addBook(ActionListener user)
+void updateBook(ActionListener user1)
+void deleteBook(ActionListener user2)
+void viewBook(ActionListener user2)
+void addUser(ActionListener user)
+void updateUser(ActionListener user1)
+void deleteUser(ActionListener user2)
+void viewUser(ActionListener user3)
+void report1(ActionListener r1)

6. Data Model

Data models define how data is connected to each other and how they are processed and stored inside the system. In other words, this concept define how a database is structured or modeled. In other words, it is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities.

The *hierarchical model* represents data as a hierarchical tree structure.

According to the XML DOM, used in this project, everything in an XML document is a **node**:

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

The XML DOM views an XML document as a tree-structure. The tree structure is called a **node-tree.** All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.

The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters).

- In a node tree, the top node is called the root
- Every node, except the root, has exactly one parent node
- A node can have any number of children
- A leaf is a node with no children
- Siblings are nodes with the same parent

```
<book id="1">

<id>1</id>

<genre>Fiction</genre>

<author>Giada De Laurentiis</author>

<year>2005</year>

<price>30.00</price>

<quantity>0.0</quantity></book>
```

In the XML above, the <title> element is the first child of the <book> element, and the <quantity> element is the last child of the <book> element. Furthermore, the <book> element is the parent node of the <title>, < author>, <year>, and <price> elements.

7. System Testing

I have tested almost every operation / method that data is entered correctly. Here are some examples:

*I tested if the password and login name introduced by two users are valid.

*I have treated the cases where there is not enough books on stock to sell them.

*I checked if the book is correctly insered or updated or removed from the xml file, in other words if the CRUD operations are made properly.

All these tests come along with error or alert messages for unfavorable cases and with information messages, plus perform the desired operation, in the favorable cases.

8. Bibliography

https://drive.google.com/drive/folders/0B-7Rn7Rec1VUaWlaVnNpUV9ROVk - Laboratory resources

http://users.utcluj.ro/~dinso/PS2017/Lectures/

https://www.w3schools.com/xml/xml_xpath.asp

https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html

https://msdn.microsoft.com/en-us/library/mt656716.aspx