<Bookstore Web Application>
Analysis and Design Document

Student: Bogdan Stupariu
Group: 30432

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 4

4. UML Sequence Diagrams 6

5. Class Design 7

6. Data Model 8

7. System Testing 8

8. Bibliography 8

1. Requirements Analysis

1.1 Assignment Specification

[Application description]

This assignment requests to design and implement an application for the front desk employees of a bookstore and has as main objective making students familiar with architectural patterns. The application should have two types of users: a regular one represented by the front desk employee whose main attribute will be to sell books, and the administrator, whose main prerogatives will be to perform operations on the regular user's database, book's database and also monitories the book's stock and user's activity.

1.2 Functional Requirements

[Present the functional requirements]

At the startup, the application should provide a login page, where users with already created accounts would log in. Because the application is only opened to the employees of a bookstore, it should not provide any registration option. After filling in the data (username and password), the login component should analyze it, and provide a proper response. If the data is invalid (by either not filling both forms or filling an invalid account) the window will show a proper message. Otherwise, the login application will redirect the user in correlation with its type. A user with administrator rights will be redirected directly to the administrator's page, while a user with regular front desk rights will be redirected to that page.

The regular user should have the following possibilities: view all the books in the store and search a specific book in the database by title, author or genre. Also, it should be capable of selling books, as a reaction of a real selling demand in a bookstore at a front desk.

The administrator will have three pages to choose in between: users, books and reports. The users page shows all the users accounts and the administrator may take CRUD actions on the accounts. All the changes are instantly saved both in the frontend and backend (stored in the XML database). The books page will also show all the information about a specific type, but this time it's about books. The administrator may perform CRUD activities on books, increase and decrease book's stock, and also search in table for a specific book (by any field). The record page shows all the sells made by the front desk users and also a generate report possibility, that generates a list with all the books that are currently out of stock. The report is available in two formats: pdf and csv.

In most of the sides of the application, the input given by the user (each type) should generate a response as output, which will be generated in tables organizing data from the database in a readable and pleasurable way.

Another important side when it comes about a banking application, will be security. It is required that at least there is no possibility of directly accessing the links without being logged in or even not accessing directly any other link than the index one (that is redirecting to the login page).

Security is assured by storing data in the browser's cache memory. By this, there is no possibility of accessing a page a user has no access to and also provides the possibility of changing/refreshing pages when the login was done successfully.

1.3 Non-functional Requirements

[Discuss the non-functional requirements for the system]

Performance requirements are directly connected with the functional side. Starting from the need of instantly selling a book when a request at a front desk is made, because obviously it is not desired to make a client wait until a page response is triggered, the page should instantly react and give a response. This aspect is assured by two aspects: firstly, the being a one-page web application, the response is dramatically faster than in other cases, and using only low time loading components is another advantage; second, the application may be run on localhost, this being a great advantage when the internet connection may be a disadvantage.

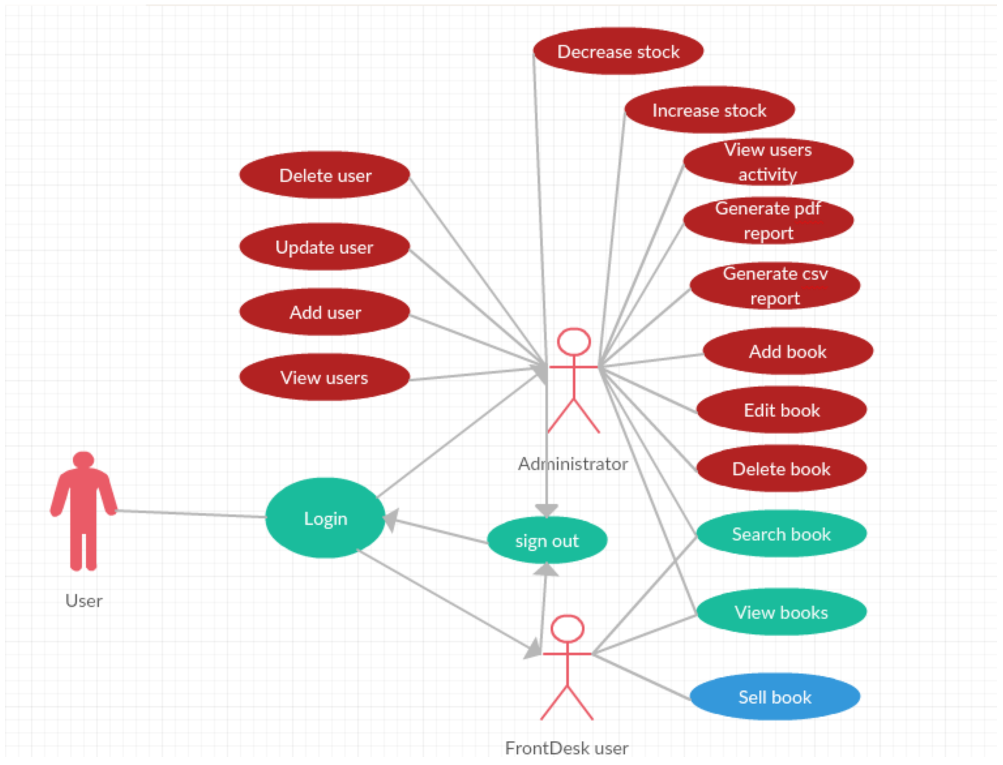
The non-functional basic requirements are the usage of the layer's architectural pattern for application organization, usage of a domain logic pattern (transaction script or domain model) / a data source hybrid pattern (table module, active record) and a data source pure pattern (table data gateway, row data gateway, data mapper). Also, all the data should be stored in multiple XML files.

Accessibility should be provided by both implementing the web application as an online application with the obvious restriction of needing an internet connection, and also giving the possibility of using the application offline, by individually installing all the components on the computer.

2. Use-Case Model

[Create the use-case diagrams and provide one use-case description (according to the format below).
Use-Case description format:
Use case: <use case goal>
Level: <one of: summary level, user-goal level, sub-function>
Primary actor: <a role name for the actor who initiates the use case>
Main success scenario: <the steps of the main success scenario from trigger to goal delivery>
Extensions: <alternate scenarios of success or failure>
]

Use case:
Decrease the number of books in stock
Level:
user-goal level – directly requested by the administrator with the goal of decreasing the number of a specific book in the database. Response is directly sent to the screen and saved in the XML file.
Primary actor:
Administrator user logged in successfully in the web application that triggers the specific button for a specific book.
Main success scenario:
The button is triggered and the data validation process begins. The amount in decreased by one and the web content is updated – the window shows the content updated (before -1). Also, at the same time, a http request is sent to the backend in order to decrease the value in the database. Value is decreased and the object is updated and saved to the XML file.
Extensions:
The data input is not well formatted and a message will appear on the desktop. May happen if the book is out of stock – number of items on stock is zero. In this case, the only response provided is an error message alerting the impossibility of doing this operation



3. System Architectural Design

3.1 Architectural Pattern Description

[Describe briefly the used architectural patterns.]

Model–View–Controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given application into three interconnected parts in order to separate internal representations of information from the ways that information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

Model - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.

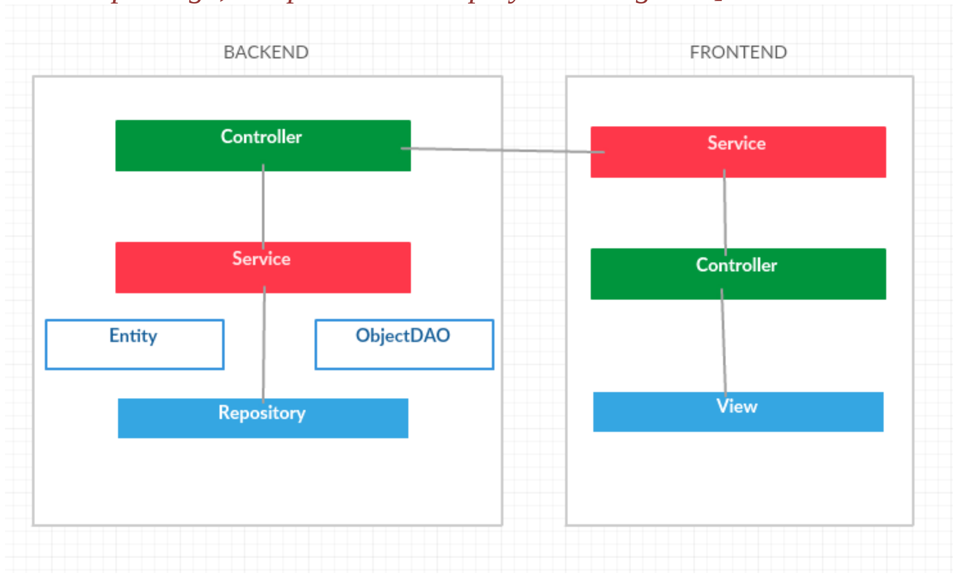
View - View represents the visualization of the data that model contains.

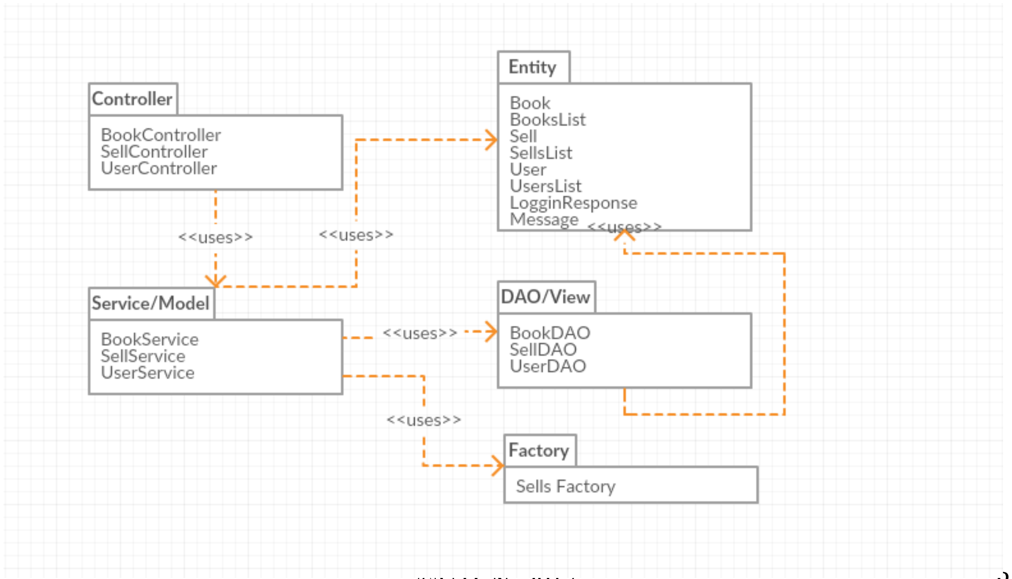
Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

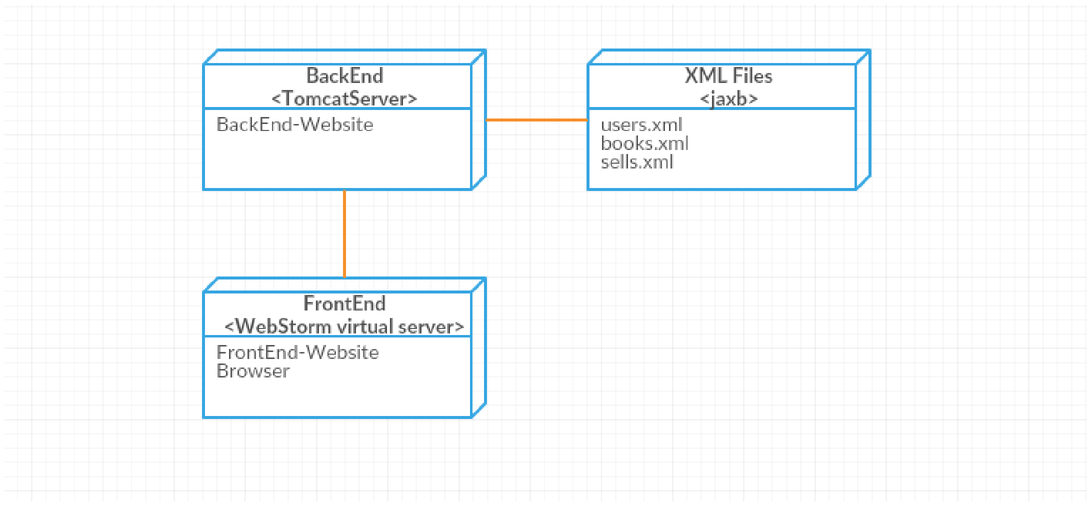
In this application, both the frontend and the backend were built according to this pattern.

3.2 Diagrams

[Create the system's conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]

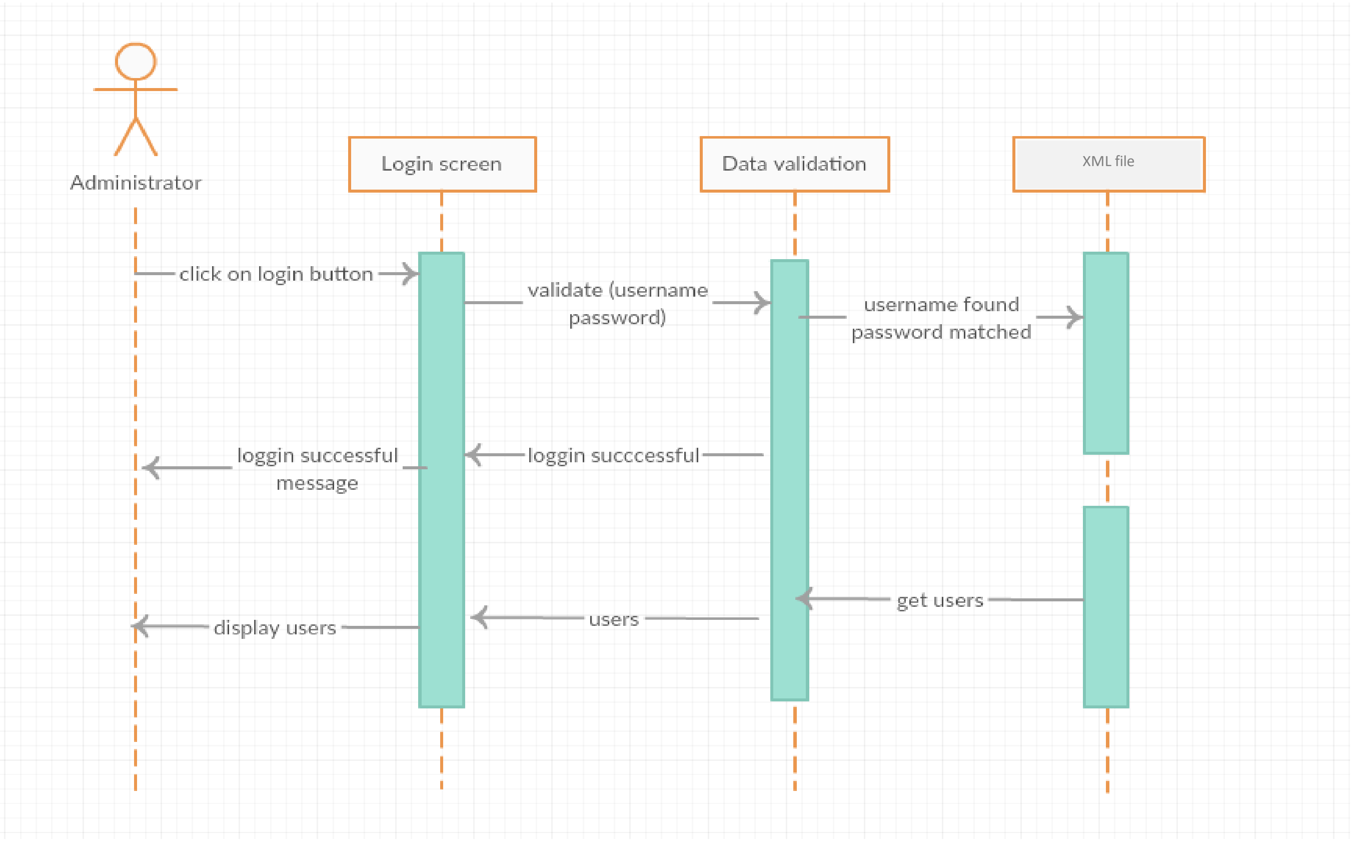






4. UML Sequence Diagrams

[Create a sequence diagram for a relevant scenario.]



5. Class Design

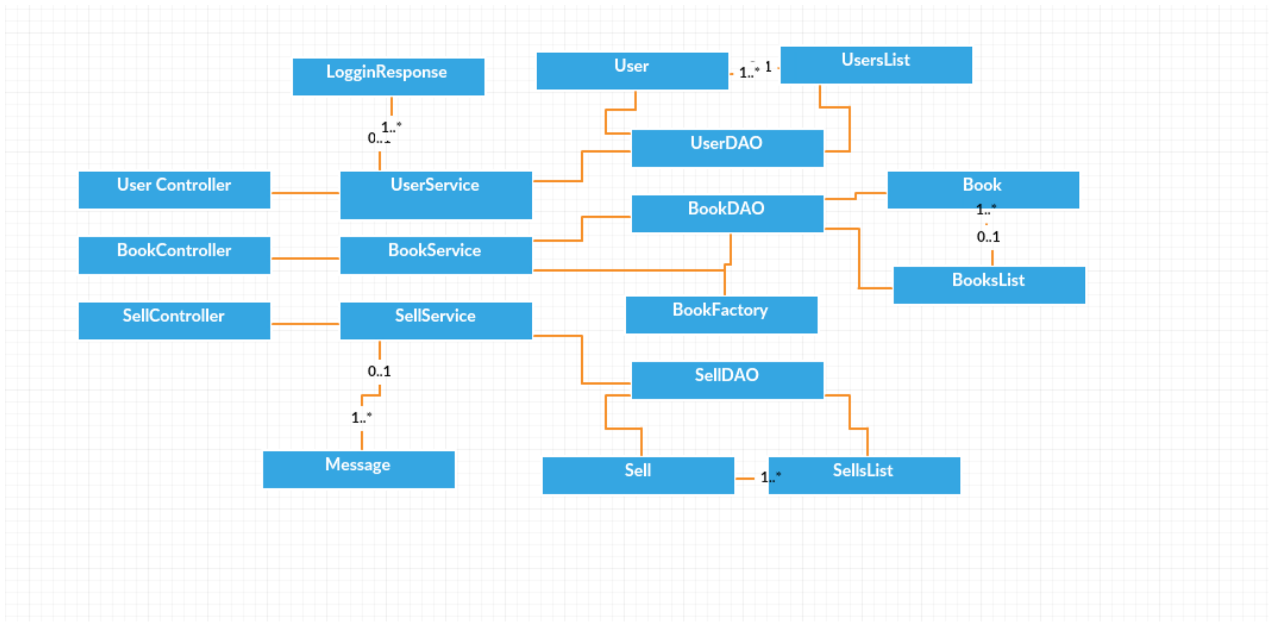
5.1 Design Patterns Description

[Describe briefly the used design patterns.]

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.
For this application, the design pattern was used for generating reports. Basically, instead of having the methods inside the View class or service/model class, a new package/class is made with the factory methods for generating reports. The methods are selecting all the books that correspond to some specific criteria – out of stock or on stock in our case.

5.2 UML Class Diagram

[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]



6. Data Model

[Present the data models used in the system's implementation.]

Data is stored in XML files: books.xml, users.xml, sells.xml.
The data saved are the following classes: Book, User, Sells, stored in ArrayLists of these types: BooksList, UsersList, SellsList.
For saving data the framework used if Jaxb.

7. System Testing

[Present the used testing strategies (unit testing, integration testing, validation testing) and testing methods (data-flow, partitioning, boundary analysis, etc.).]

Data flow testing is a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of variables or data objects. Dataflow Testing focuses on the points at which variables receive values and the points at which these values are used.
This type of testing was very relevant because of the usage of http communication. The objects sent were tested in frontend against the input data, in the frontend before submitting the http message, in the backend when receiving http message and in the backend before storing it to the XML databases. This data flow testing method made the data transfer valid.

Validation testing. All input data is tested against invalid data by using regex data validation. Regex Pattern are supposed to be matching the input data and so to validate the input. Otherwise, the input will be rejected and a message will occupy.
Examples of patterns:

```
$scope.isbnPattern = /^[0-9]{1}[-]{1}[0-9]{3}[-]{1}[0-9]{5}[-]{1}[0-9]{0,1}$/i;
$scope.authorsPattern = /^[a-zA-Z]+(([\'\,\.\- ][a-zA-Z ])?[a-zA-Z]*)*$/i;
$scope.yearPattern = /^([1]?|[2]?)[0-9]?[0-9]?[0-9]$/i;
$scope.imagePattern = /^[h][t][t][p][s]?[:][\/][\/]([a-zA-Z]|[0-
9]?)+[.][\S+]+[\/][\S+]+[.](([j][p][e][g])|([j][p][g])|([p][n][g])|([g][i][f])|([b][m]
[p])|([t][i][f][f]))$/i;
$scope.pricePattern = /^([1-9]+)([0-9]+)?$/i;
```

8. Bibliography

1. http://stackoverflow.com/
2. https://docs.angularjs.org/api
3. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
4. http://getbootstrap.com/components/
5. https://www.codecademy.com/learn/javascript
6. Youtube tutorials