<Assignment 2>

Analysis and Design Document

Student: Ștefănescu Marian

**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.

# 1.2 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 application must comply in terms of:

* Scalability, in order to support a large number of users and books
* Reliability, as any errors will be handled locally and they should not affect the general performance at runtime.
* Data integrity, because the information stored is of high importance
* Security, because only the authorized users must be able to enter the application.

2. Use-Case Model

Use case: Accessing the members dashboard

Level: summary-level

Primary actor: Customer

Main success scenario: The customer accesses the application, logs in with the correct credentials (otherwise it will be redirected back to the login screen) and reaches at the index page. The navigation bar is updated to reflect the fact the user is logged in, and now he has access to new menu entries like View all books and Single Book. From the Single Book page he can click on Buy and his cart will get updated with the new information.

Extensions: In the case his username and password correspond to admin credentials the user will be granted administrator rights. As administrator, he can edit all the information about the books, add or delete new ones and generate reports for the books that aren’t currently in stock.

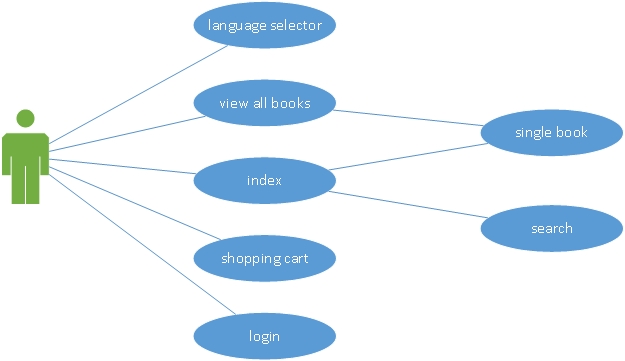


Figure Use case for the normal user

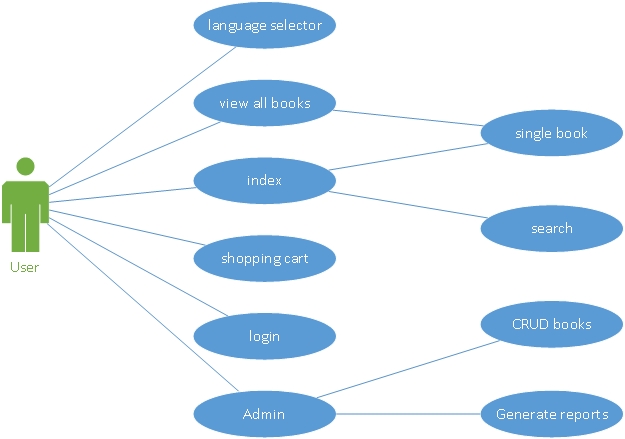


Figure Use case for the administrator

3. System Architectural Design

**3.1 Architectural Pattern Description**

I’ve used the Model View Controller pattern because it molds very well on the general structure of a web application. Website are inherently developed around this architecture because, at the simplest level, the browser works as a Controller, the HTML as a Model, and CSS as a view. This simple idea is extended though, and the model becomes a set of specialized classes that can parse XML files properly and create an Object Model from them.

A birds-eye view of my application will highlight a series of repetitive patterns that follow the MVC pattern. Each component of the website that I’ve considered to have the potential (or the one that already has) has been assigned a specific package(book, login, index etc.). Every package follows the MVC architecture, but some components have less complexity so a Model and View may lack entirely (IndexController for example).

In my application, the Model is a little different in comparison with the default case, as I’ve created concrete classes for every type of object that might be found in the actual application. The idea is to use them in conjunction with the lists obtained from the DOM classes and to process the data using the basic getters/setters of the simpler class.

A single package doesn’t follow the MVC approach and that is the document package that contains the factory methods that help build the pdf and csv reports. The Factory Pattern is a natural and powerful way to use polymorphism in your advantage as you create a general interface that is used by a Factory Creator class. In the factory class, a static methods generates, by using a String, the correspondent subclass. The idea is to transfer the details of the concrete implementation to the subclasses, but without having the know the specific details of each one.

In my example, the factory class generates an instance of the subclasses PDF and CVS. Each of these classes contain a method **create()** that takes as input a list of books which is supplied by the Book Model.

**3.2 Diagrams**

Controller

Model

View

Figure Structura generala a unui pachet

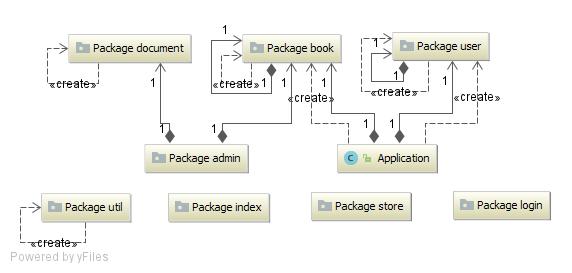


Figure 4 Dependentele intre pachete

Client

Server

Model Data classifiers

Model

Controllers



XML

Figure Structura arhitecturala a sistemului

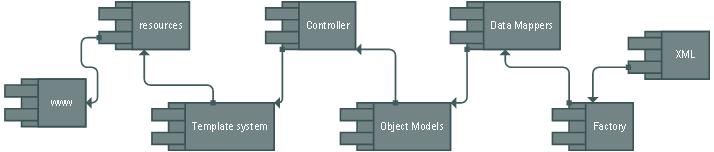


Figure Diagrama de componente

4. UML Sequence Diagrams

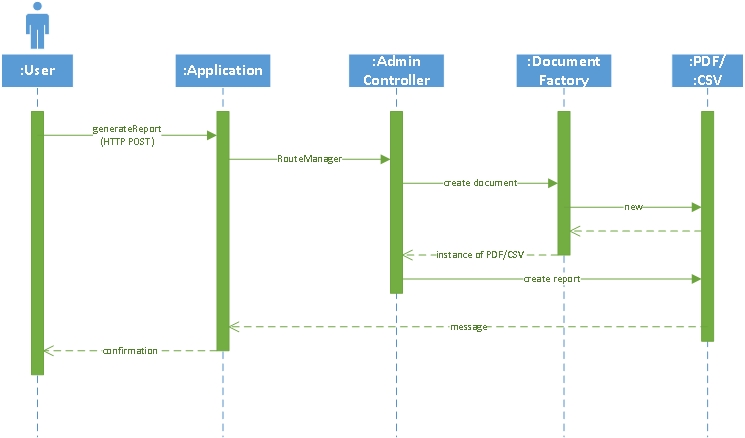


Figure 7 Sequence diagram for generating a report

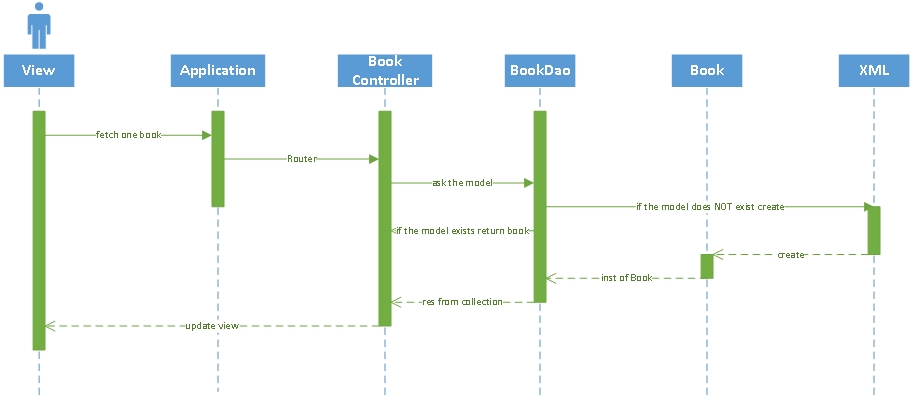


Figure 8 Sequence diagram for viewing one book

5. Class Design

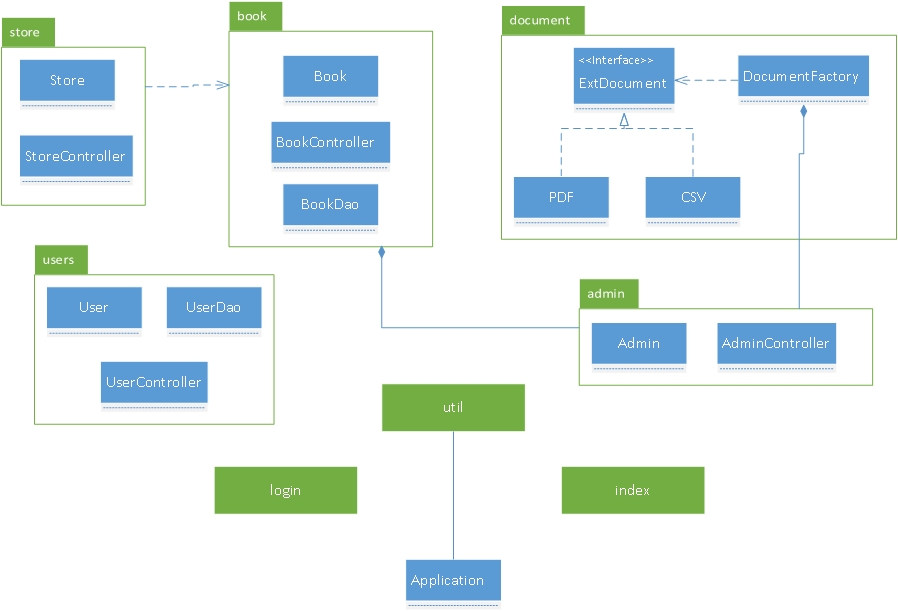
**5.1 Design Patterns Description**

1. The system as a whole uses a collection of micro-MVC that are stored in packages separately.
2. At the View level, I use a simple Template View pattern because I have a collection of static HTML pages and the dynamic content parts are inserted using a Template Framework called Velocity. This does a pre parse of the HTML code and searches for special template tags which are replaced with the content fetched from thru the business layer.

**5.2 UML Class Diagram**

I’ve chosen the structure shown above for this assignment as it follows the MVC pattern. The general structure isn’t a pure MVC because some components are better not having a certain degree of rigidity the MVC pattern imposes. All in all, the classes that are directly related to the business logic of the application follow the MVC pattern rigorously.

Not shown in the diagram is the Template View pattern I used with the Velocity Framework, which is dependent only from the classes that compose the Controllers and the utility classes, along with the main application class that set the get and post routes of the website. I find this important, as it reduces the interference between the java dynamic content and the static HTML to a minimum.



6. Data Model

For this assignment, I’ve used a non relational data storage mechanism. Java has great support for XML manipulation, so all the website’s data is stored in two external .xml files, called **users** and **books**.

At the implementation level I use two different techniques to access the content of the .xml files. First is by using XPath, with which I can access very easily all the data in the .xml files with query like strings. Those query type strings are the equivalent of SELECT statements from MySQL. XPath can be used though only to access data. To effectively change, add or delete the data, I first update all nodes text content using the Java classes with the same name and then, by using the Transformer and TransformerFactory classes I effectively update the new node structure too.



Figure 9 XML structure of users file

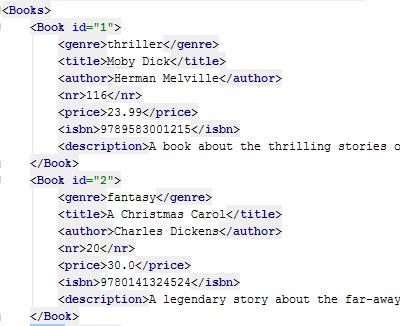


Figure 10 XML structure of books file

7. System Testing

Because I don’t have a large scale system, I decided to skip unit testing, as I was pretty confident in the individual units of the application.

During the entire time of development, I’ve done integration testing in a top down manner, because it just so happened I had the high level modules developed first. This was an advantage in the sense I was more than confident in the routers and the methods that handled data transmission thru the services the presentation layer provided. The major downside however was that low level utilities were tested last and unfortunately, due to some code mistakes, I’ve found to be very hard and quite time consuming to find the sources of the data inconsistencies.

8. Bibliography

* Fowler, Martin (2002). Patterns of Enterprise Application Architecture. Addison-Wesley
* <https://msdn.microsoft.com/en-us/library/ff649643.aspx>
* <https://www.javatpoint.com/MVC-in-jsp>
* <https://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture>