Assignment 2

Student: Vereș Adela

**Group: 30235**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 4

2. Use-Case Model 4

3. System Architectural Design 5

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 9

7. System Testing 10

8. Bibliography 10

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.

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.

# Functional Requirements

* Ability to authenticate either as an employee, or as an admin, using a username and a password, via the login page.
* Ability to do the following, while logged in as an employee:

- access only the login page or stay on the current page

- view the list of existing registered books, and their details

- search books by title, author, and genre

- sell a number of copies of an existing book

* Ability to do the following, while logged in as an admin:

- access only the login page, the employee page, and the books page

- on the employee page:

- view the list of existing employees, and their details

- update a selected employee

- delete a selected employee

- add a new employee - only possible if introducing a personal numerical code different from the existing ones.

- on the books page:

- view the list of existing books, and their details

- update a selected book

- delete a selected book

- add a new book – only possible if both title and author differ from any of the existing books

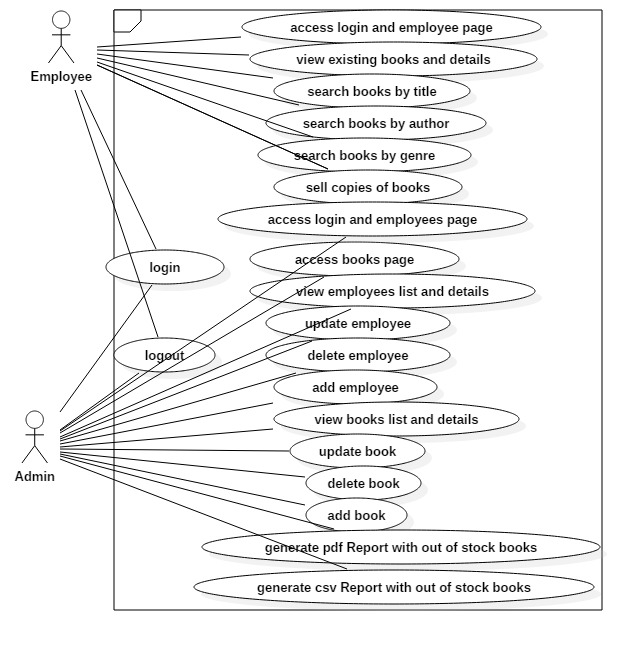
- generate .pdf and .csv reports with the out-of-stock books

# Non-functional Requirements

The system presents the following quality attributes:

* *Usability*: the software product can be used by specific consumers – employees and the administrator – in order to facilitate easy, effective access to the application. This is enabled due to the use of user-friendly, intuitive graphical user interface. The layout of the webpage is clearly designed in order to support all operations, while the server side implementation renders qualitative response time.
* In terms of *performance*, the system is defined by a satisfactory *response time,* low *latency,* satisfactory utilisation of *computing resources.*
* *Reusability* and *extensibility* are supported due to the use of *low coupling* and separated functionalities that comes with the implementation of the *MVC* architectural pattern.
* *Scalability* is supported as well, being possible to handle large amounts of data – high number of clients, accounts, employees.

2. Use-Case Model



Use case: add book

Level: admin-goal level

Primary actor: admin

Main success scenario:

* login as admin
* from the employee page, go to the books page, by clicking on the link in the navigation bar
* on the books page, fill the form field with the new book’s details
* after having finished, select the ‘Add’ button
* view the list of the books, and locate the newly inserted book

**Extensions:**

* after having filled the form, upon clicking the ‘Add’ button, if the title and author match the ones of an existing book, the action will result in failure, an error message being displayed that there exists a book with identical title and author.
* the ‘Add’ button will only be available for clicking after having correctly filled each field – the fields are checked for inputs such as integer-for the ‘quantity’ field- , double/float – for the ‘price’ field- and respectively, strings- for the rest of the fields, with a minimu length of 3 characters.

3. System Architectural Design

**3.1 Architectural Pattern Description**

The project follows the MVC pattern, implemented using the Spring Boot framework, while also incorporating the Factory Method design pattern, as explained below. The classes are organised in packages, being able to be grouped as follows:

● **The Model**: is represented by the *model* package which contains the following classes:

- Book, BookStore, Employee, Employees, LoginAccount, LoginAccounts – these are all mappings from the ‘.xml’ files containing specific data, with the help of the specific xml annotations from java;

- BooksList, EmployeesList, LoginAccountsList – classes that marshall and unmarshall from object to xml representation, with the help of Jaxb Library.

The Model also comprises the business logic, thus the *service* package is included here, with the BookService, EmployeeService, LoginService interfaces and implementations, containing the methods that handle data information.

The service package also contains the Report interface and PdfReport, CsvReport, and ReportFactory classes, which, implementing the **Factory Method design pattern**, help in constructing the desired report, and write the out-of-stock books list to the specific files on the server.

●**The Controller**: is represented by the *controller* package containing the AppController and RestApiController classes, handling the incoming client requests, and calling the specific methods in the *service* package to fulfill the requests.

The main method, responsible with bootstraping the application, is in the

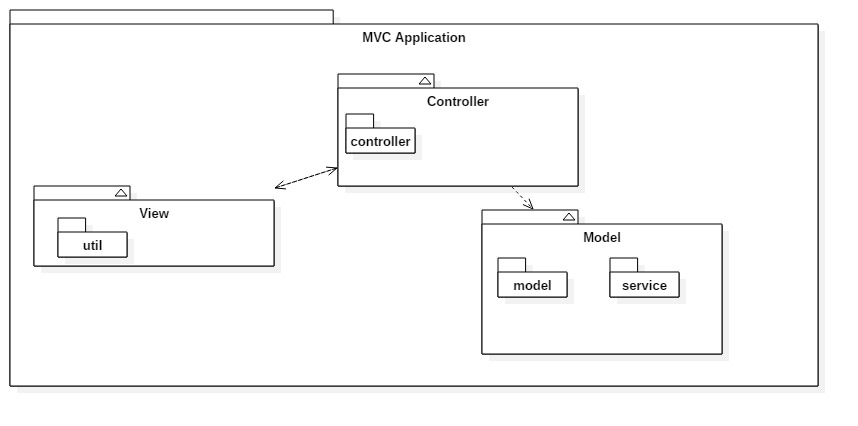
main package of the application, in the BookStoreApp class.

●**The View** of the application is represented by the user-interface, created with the help of ‘.ftl’ pages, JavaScript files, using AngularJS, and styling elements (Bootstrap elements included).

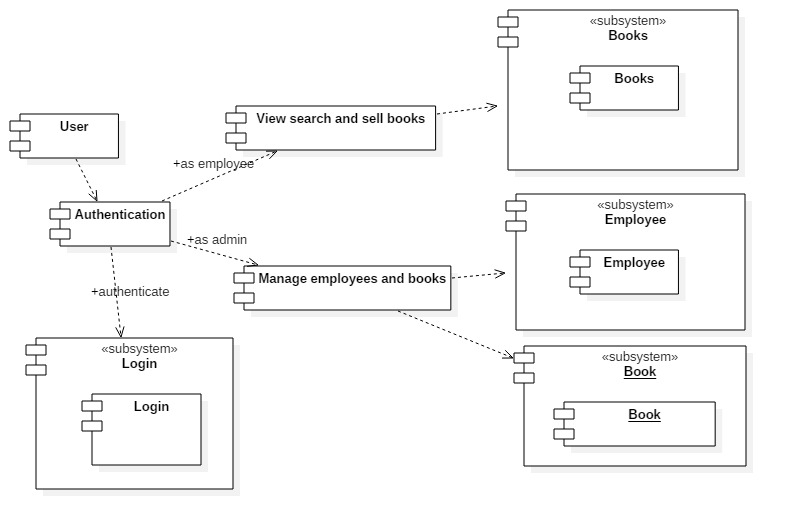
The *util* package handles the appearance of possible errors, with the CustomErrorType class.

**3.2 Diagrams**

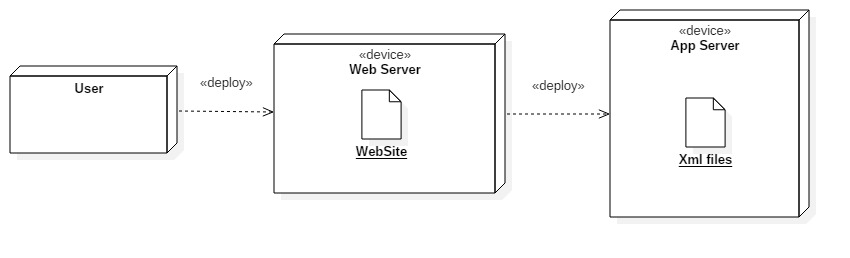
● Package diagram



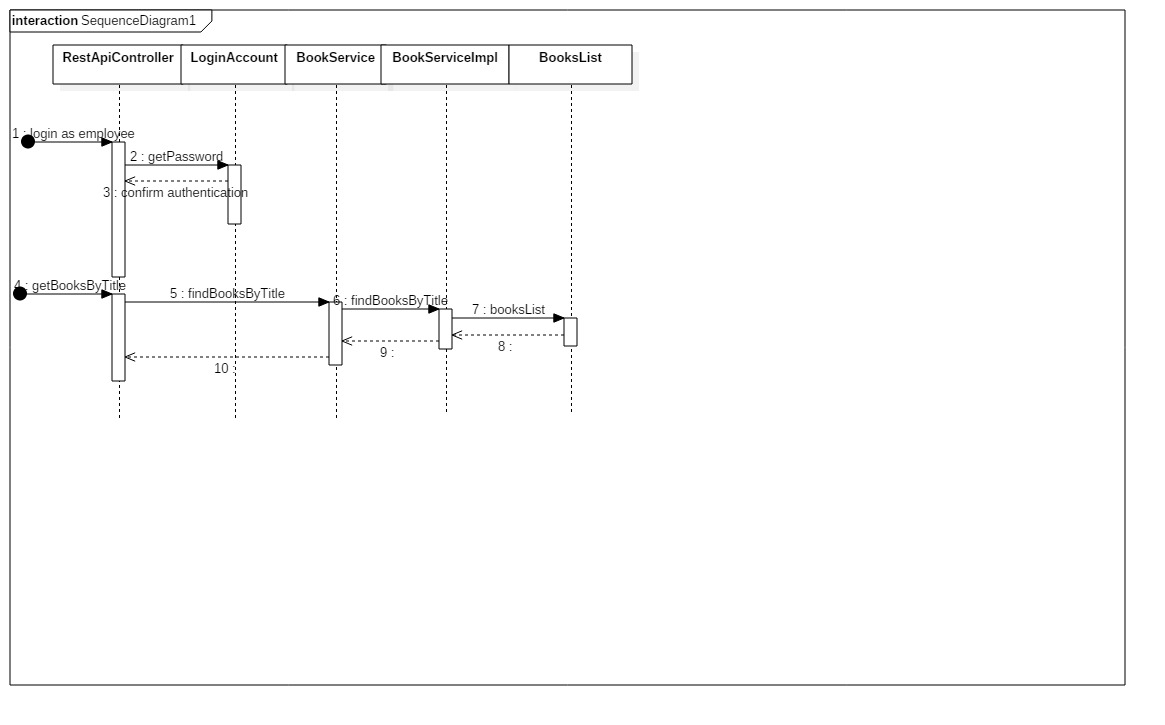
● Component diagram



● Deployment diagram



4. UML Sequence Diagrams



5. Class Design

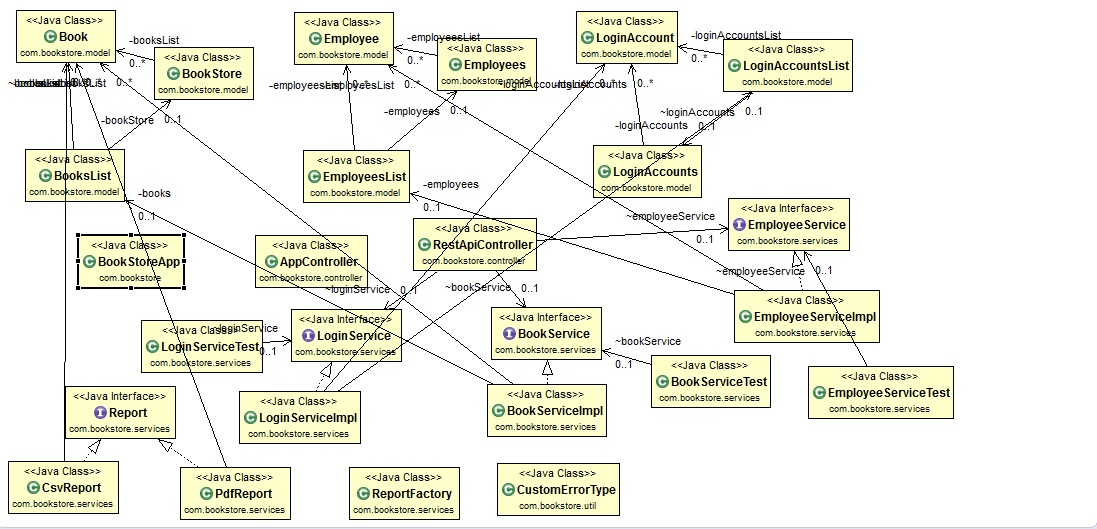
**5.1 Design Patterns Description**

The used design patterns are  **MVC** and **Factory Method.**

The MVC design pattern is modelled using the SpringBoot framework, consisting in ‘controller’ packages (containing the actual *control* package) that interract with data, and communicate the changes to the view. The ‘model’ package here contains packages: *services,* and *model.* These packages perform operations on data and retrieve it from the xml files. The ‘view’ is represented by the user-interface. The AngularJS JavaScript framework is used to model the dynamic behaviour of the webpage, as well as bind the application data to html format. The interface is modelled using Html, Css and Bootstrap elements.

Factory Method design pattern is used to delegate the creation of concrete classes instances, to subclasses rather than the calling class, with the help of a constructor method, from a constructor class. This method is passed a string as an input parameter, used as a decision factor in the creation of those classes. For this project, the interface Report was created, with the method generateReport() that is implemented in the two distinct classes created as well: PdfReport and CsvReport. The ReportFactory class contains the getReport method, with a string as the report type to be chosen, passed in as a parameter. This method decides which object to create and return afterwards. The actual calling of those methods takes place in the getCsvFile and getPdfFile from the RestApiController class, where a factory object is created, and then the getReport method is called with the apropiate string as the incoming parameter – here: ‘csv’ or ‘pdf’. Next the report is generated by calling the generateReport method.

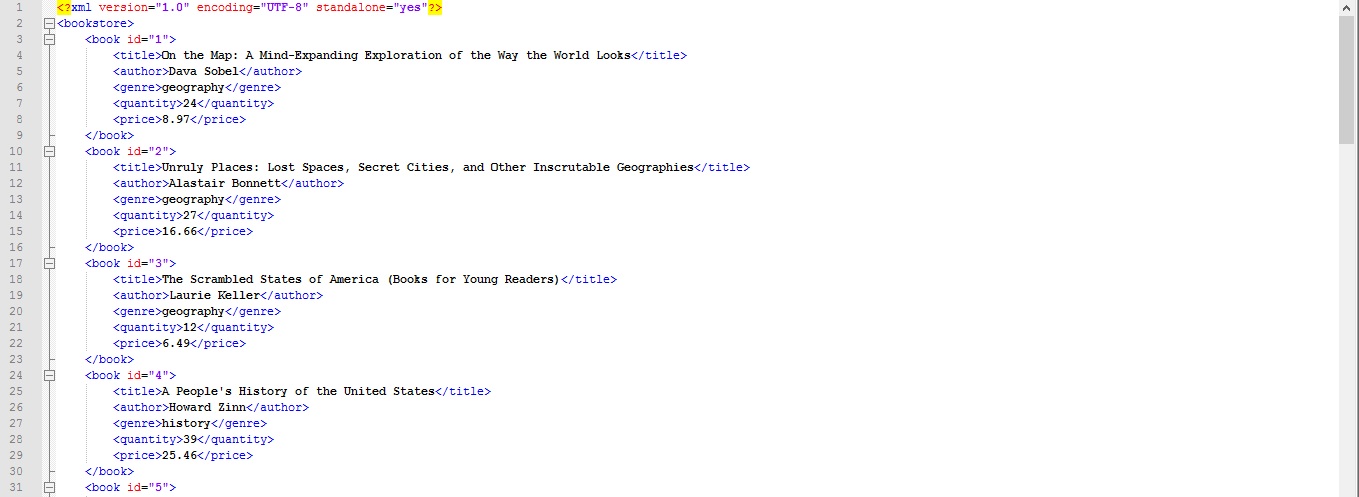
**5.2 UML Class Diagram**



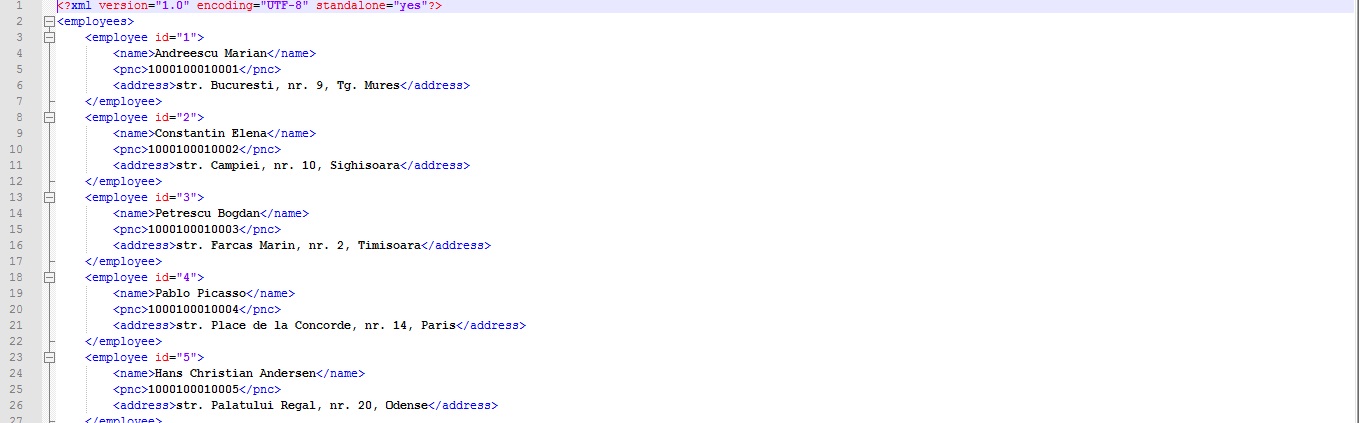
6. Data Model

The data model is represented with the help of ‘.xml’ files. There are three such files, each storing information about: login accounts, books, employees. The three snapshots that follow capture this information:

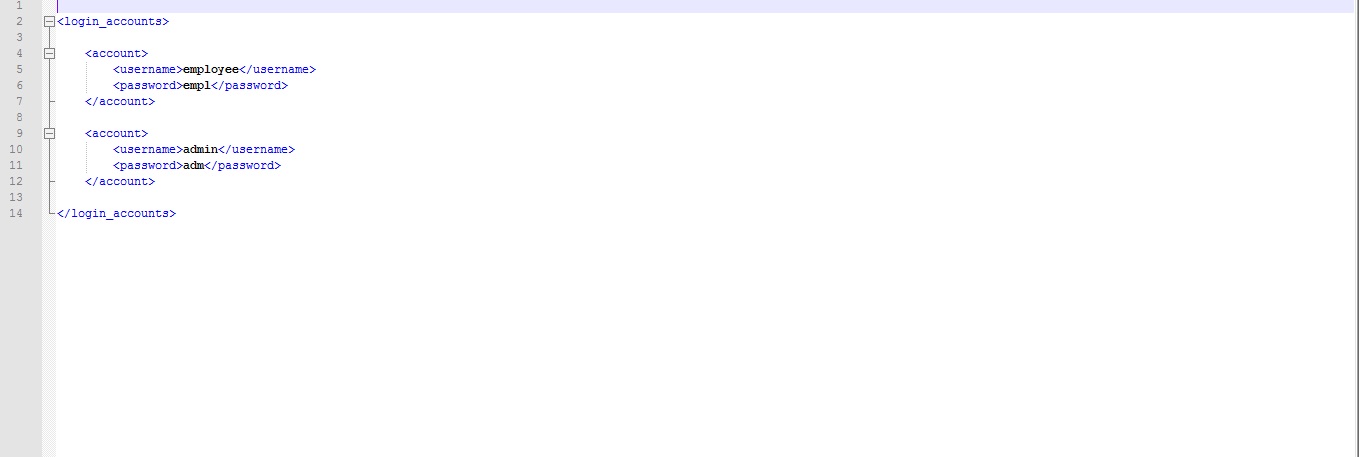
-books:



* employees:



* login:



7. System Testing

The system includes unit testing classes for unit testing. The Junit4 framework is used, and tests are methods encapsulated in a single class for each of the corresponding services classes. Thus we have: EmployeeServiceTest, BookServiceTest and LoginServiceTest – classes testing each method provided by the correspondent service classes. Methods such as ‘findBookById, addEmploye, deleteBook, etc, are tested, thus testing in fact the CRUD operations’ well-functioning.

8. Bibliography

1. Spring Framework:  
<https://en.wikipedia.org/wiki/Spring_Framework>

# 2. AngularJS tutorial: <https://www.w3schools.com/angular/default.asp>

3. Martin Fowler, Patterns of Enterprise Application Architecture:  
<http://disi.unal.edu.co/dacursci/sistemasycomputacion/docs/SWEBOK/Systems%20Engineering%20-%20EAA%20-%20Patterns%20of%20Enterprise%20Application%20Architecture%20-%20Addison%20Wesley.pdf>

# 4. Spring Boot, AngularJS, Spring Data, JPA CRUD App Example:

<http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/>

# 5. Http status codes: <https://httpstatuses.com/>