<BookStore>

Student:Cimpan Roxana-Adriana

**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.

# 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).

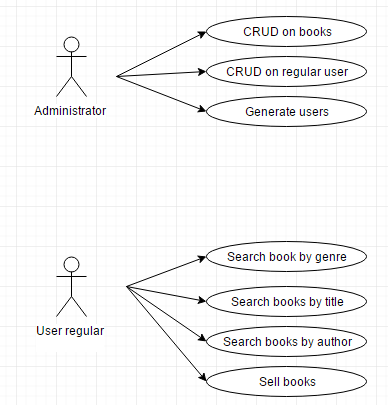
- CRUD on regular users’ information.

- Generate two types of reports files, one in pdf format and one in csv format, with thebooks out of stock.

# Non-functional Requirements

-generate reports

2. Use-Case Model



**Use case:**Log In

**Level:**user

**Primary actor:**Administratorul se logheaza

**Main success scenario:** Se completeaza field-urile corespunzatoare pentru logare: nume si parola. Ca urmare a acestei actiuni se va deschide o fereastra prin intermediul careia administratorul va putea sa-si realizeze operatiile specifice.

Extensions: Trebuie introduse corect, atat numele, cat si parola, altfel se va afisa un mesaj de eroare si nu va putea fi accesata acea fereastra.

3. System Architectural Design

**3.1 Architectural Pattern Description**

**Modelul** reprezinta partea de hard-programming, partea logica a aplicatiei. El are in responsabilitate actiunile si operatiile asupra datelor, autentificarea utilizatorilor, integrarea diverselor clase ce permit procesarea informatiilor din diverse baze de date.

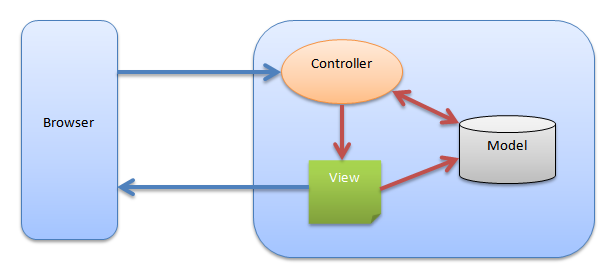
**View**-ul se ocupa de afisarea datelor, practic aceasta parte a programului va avea grija de cum vede end-userul informatia procesata de controller. O data ce functiile sunt executate de model, viewului ii sunt oferite rezultatele, iar acesta le va trimite catre browser. In general viewul este o mini-aplicatie ce ajuta la randarea unor informatii, avand la baza diverse template-uri.

**Controller**-ul reprezinta creierul aplicatiei. Aceasta face legatura intre model si view, intre actiunile userului si partea decizionala a aplicatiei. In functie de nevoile utilizatorului, controllerul apeleaza diverse functii definite special pentru sectiunea de site in care se afla userul. Functia se va folosi de model pentru a prelucra (extrage, actualiza) datele, dupa care informatiile noi vor fi trimise catre view, ce le va afisa apoi prin template-uri.

In pachetul controller am implementat clasele:

* + - AuthorController
    - BookController
    - UserController
    - EmployeeController

**3.2 Diagrams**



In pachetul model am implementat clasele:

* + - Author
    - Book
    - User
    - Employee

Partea de view in aplicatia pe care am implementat-o o reprezinta html-urile

In pachetul controller am implementat clasele:

* + - AuthorController
    - BookController
    - UserController
    - EmployeeController

4. UML Sequence Diagrams

5. Class Design

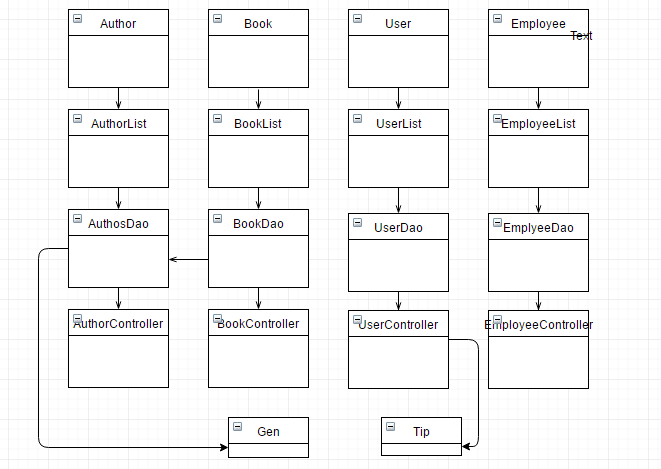
**5.1 Design Patterns Description**

* FACTORY:

-implementez o interfata IRaport

-implementarea urma sa o fac pentru generarea de rapoarte xml si csv

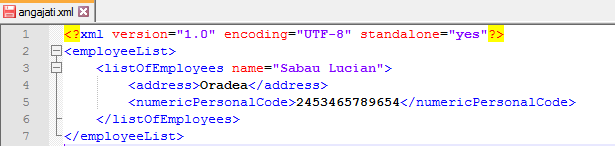
**5.2 UML Class Diagram**



6. Data Model

-fisiere xml:

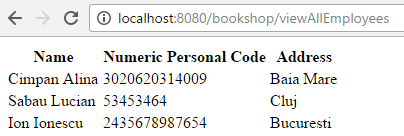
* autori.xml



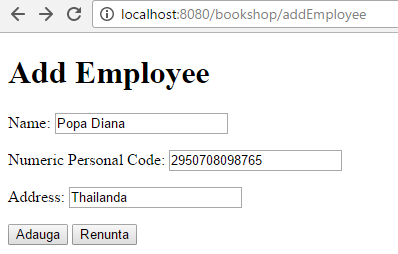
* angajati.xml
* carti.xml

7. System Testing

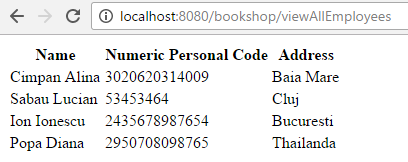
-pentru a vizualiza angajatii: http://localhost:8080/bookshop/viewAllEmployees



-pentru a adauda angajati: http://localhost:8080/bookshop/addEmployee



-observam ca daca accesam iar <http://localhost:8080/bookshop/addEmployee> angajatul este adaugat cu succes



8. Bibliography

<http://howtodoinjava.com/spring/spring-mvc/spring-mvc-requestmapping-annotation-examples/>

<http://www.javawebtutor.com/articles/spring/spring-mvc-requestmapping-example.php>

<https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html>

<https://spring.io/guides/gs/serving-web-content/>

<http://howtodoinjava.com/jaxb/jaxb-exmaple-marshalling-and-unmarshalling-list-or-set-of-objects/>

<https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/twbs_jaxbmarshalxml.html>

<http://stackoverflow.com/questions/10534126/need-help-in-formatting-jaxb-output>

<http://stackoverflow.com/questions/17987380/combine-get-and-post-request-methods-in-spring>

<http://stackoverflow.com/questions/14254049/spring-framework-difference-between-get-and-post>

<http://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib-method-args>

<https://docs.google.com/spreadsheets/d/1hszU7u7XchVOGi4dHwn0rguPnGwFswQwZSeA_lJMEL0/pub?hl=en_US&gid=2>

<https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm>