Technical University of Cluj-Napoca Software Design 2020-2021

Book Store Application

Assignment 2

Hunor Debreczeni Group 30433

Contents

1	Requirements		3
	1.1	Assignment specification	
	1.2	Functional requirements	3
	1.3	Non-functional requirements	3
2	Use	e case model	4
3	Sys	tem architectural design	4
4	UN	IL Sequence diagrams	5
	4.1	Modify stock in Cart	5
	4.2	Finalize order	6
	4.3	User data persistence	7
5	Cla	ss design	8
6	Dat	a Model	9
7	Sys	System testing	
8	Bib	Bibliography2	

1 REQUIREMENTS

1.1 ASSIGNMENT SPECIFICATION

Use Java/C# API to design and implement an application for the employees of a bookstore. The application should have two types of users (a regular user represented by the bookstore employee and an administrator user) which must provide a username and a password in order to use the application.

1.2 FUNCTIONAL REQUIREMENTS

There should be two types of users, the seller and the manager.

The **Seller** can perform the following operations:

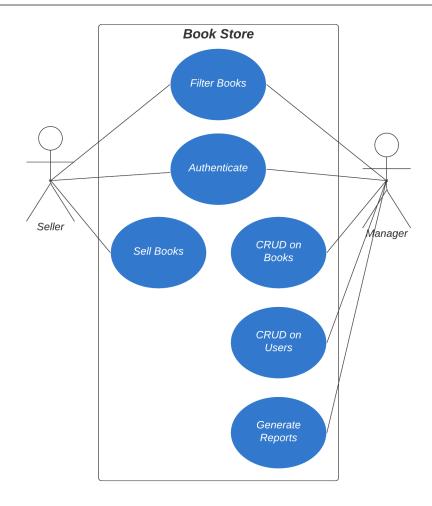
- Search books by genre, title, author.
- Sell books.

The **Manager** can perform the following operations:

- CRUD on books (book information: title, author, genre, quantity, and price).
- CRUD on regular users' information.

1.3 Non-functional requirements

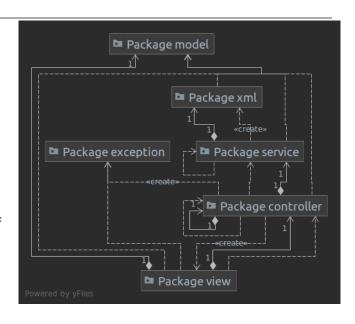
- The information about users, books and selling will be stored in multiple XML files. Use the Model View Controller in designing the application. Use the Factory Method design pattern for generating the reports.
- All the inputs of the application will be **validated** against invalid data before submitting the data and saving it.



3 SYSTEM ARCHITECTURAL DESIGN

The system used the **MVC** (Model – View – Controller) layered architecture. It consists of a data model, the presentation information and some controllers for the information. There's still need for the service layer which handles the data persistence.

The **View** package contains the part of the application, which can be seen by the end-user (seller or manager in our case). Each view object uses the controller, to modify the current state of the application. This view also uses the model package, to transfer the data between the packages.



The **Controller** package handles all the logic from the application. It controls every view, and also the data persistence which may be modified by the end-user, through the view. For validation, exceptions are raised, to make the handling easier, by alerting the view that something is wrong.

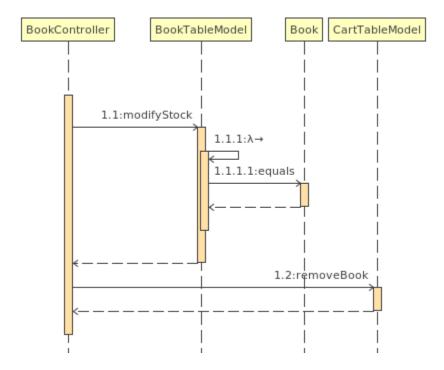
The **Service and XML** package is used for the data persistence. It handles the reading/ writing of the data to multiple xml files, which represent the database in the current application.

The **Model** package contains all the information for the objects that are needed, such as User, Book, Access Type, Reports, Lists and Tables.

4 UML SEQUENCE DIAGRAMS

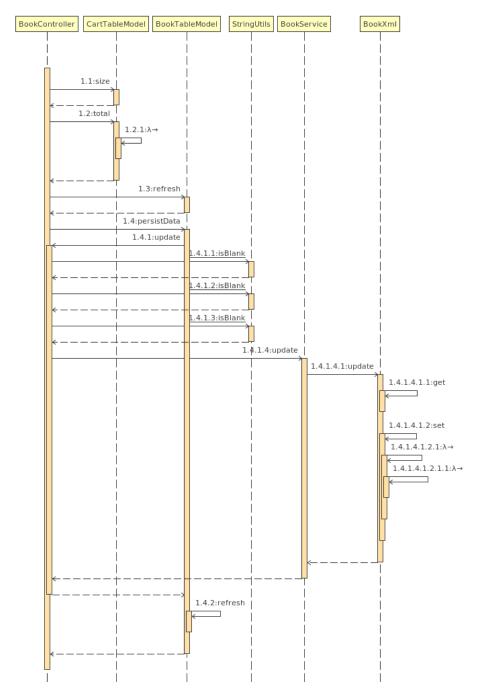
4.1 Modify Stock in Cart

The following seq. diagram describes the flow on the event of modifying the quantity of a given book in the cart. It verifies first if there are enough books in stock of a given book. If there is, then the cart is modified successfully. If the new stock is negative or equal with zero, then the book is removed from the cart.



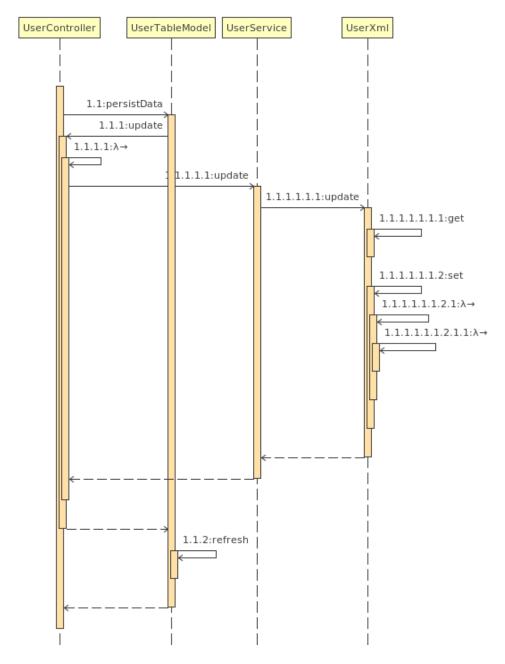
4.2 FINALIZE ORDER

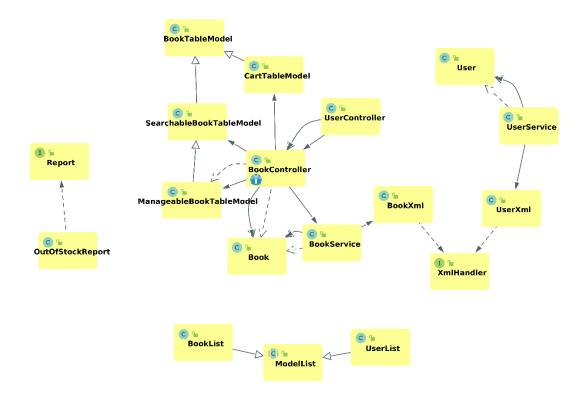
This flow verifies if the cart contains books first. If it contains, then we get the total price. After that the cart is refreshed/ cleared. To save the quantity of the books/ the stock, the data from the book table must be persisted. For each book, an update action is made, which also verifies if the fields are correct. As a last step, the data from the original XML file is taken and replaced with the new one. This is done, to ensure that we don't modify data that shouldn't be modified.



4.3 USER DATA PERSISTENCE

This flow is similar to the persistence of the books described above. The data is taken from the UserTableModel and for each User, the data is updated to ensure that no data is lost.





The brain of the application is the BookController. It handles every request, with the exception of the ones which are intended for modifying a User model. Even the UserController uses a BookController upon login, to decide the homepage of the given user (Manager Dashboard/ Seller Dashboard).

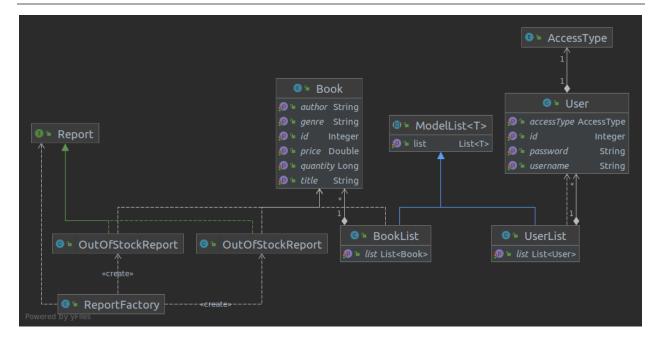
It can also be seen, that for displaying the Books, multiple Table Models were created, which add new functionality to it. The super class, BookTableModel only defines the handling of the main data (list of books). The Searchable Tabel Model adds filtering functionality, the Manageable Book Table Model adds editing functionality. The Cart Table Model adds cart functionality: editing stock, adding items, which increment the quantity of a given book if it's already in the cart.

We can also see the Report classes, where at the moment is only a single type of Report, the Out of Stock report. It comes in two different output methods (PDF/CSV), but they have the same name, but in different packages.

The service classes can also be seen which in the end, use the XmlHandler classes, which implement the main methods (CRUD) on xml files with the specified type.

The ModelList class is used, to make it easier to output the Books and the Users in an xml file.

6 DATA MODEL



The application consists of three main data models. The Book, User and Report models. The application is made for a book store, thus the most important model is the Book.

The Book model contains information about books, and every field must be populated upon creation, with the exception of the ID field, which is automatically populated by using the XmlHandlers from the back.

The User model is used for authentication and permission differentiations. This model also uses the Access Type enumeration class, which contains objects that represent different kind of users for the application. In our case, the Manager and the Seller.

The Report contains books, that have zero quantity in our case. This may also be upgraded in the future, to contain different kind of reports also.

The UserList and the BookList class is used for handling the XML files in an easier way.

7 System testing

For testing the System, I went with the manual testing approach, doing the task and verifying if the result is the expected one.

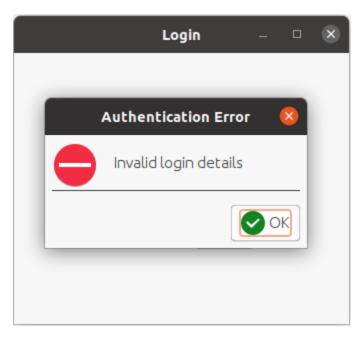


Figure 1 Authentication Verification

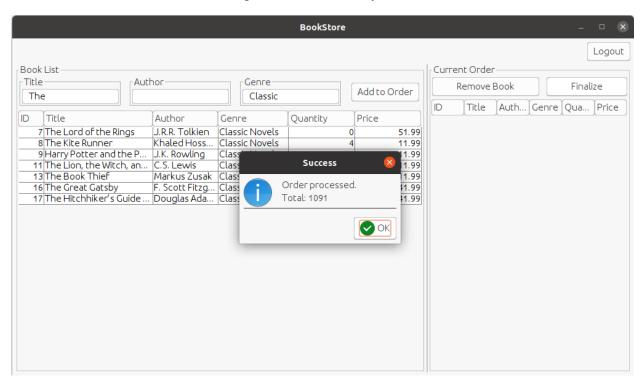


Figure 2 Cart processing upon correct cart content

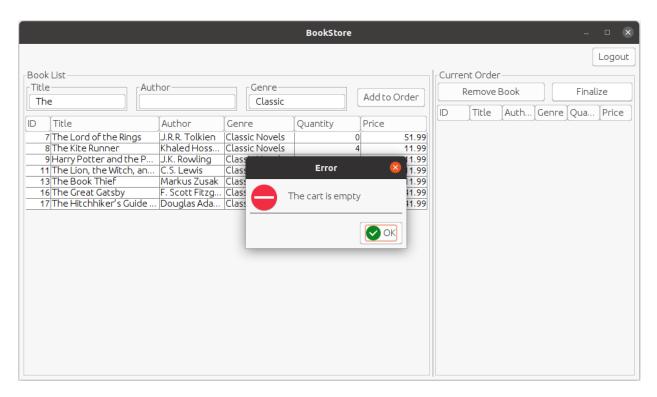


Figure 3 Finalize empty cart error

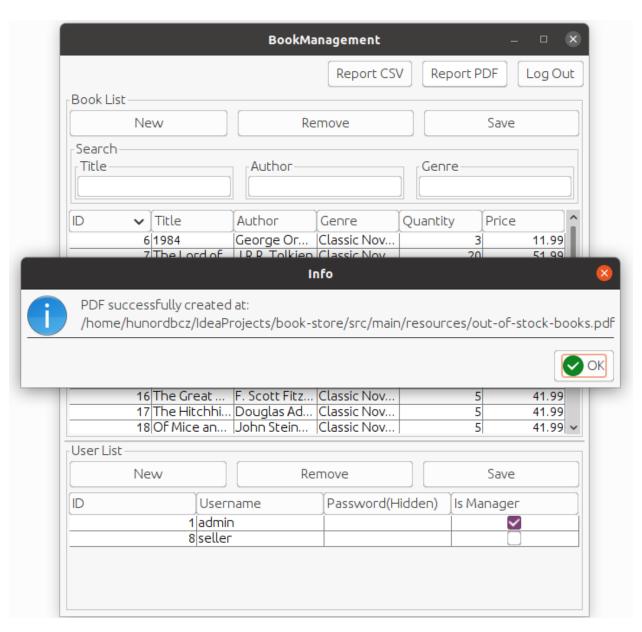


Figure 4 Generate PDF report

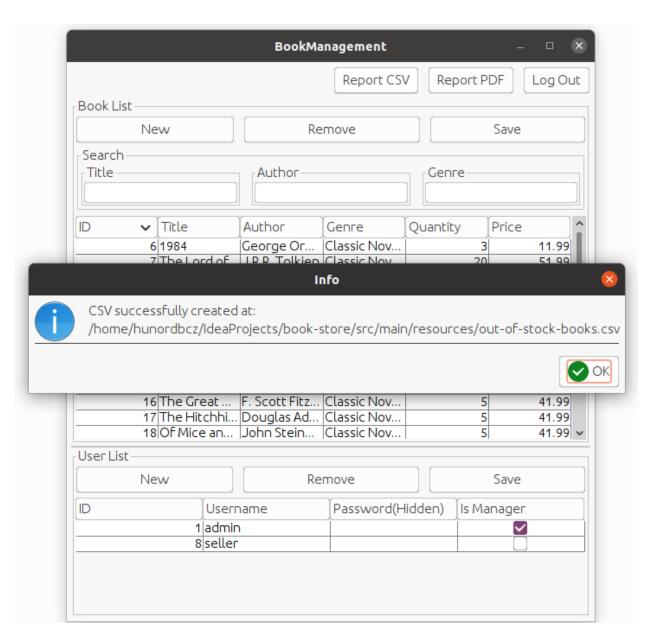


Figure 5 Generate CSV report

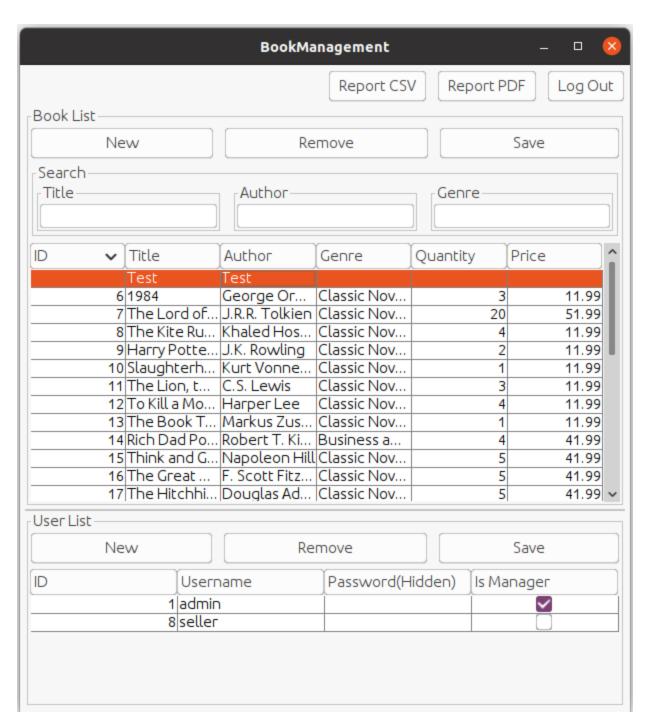


Figure 6 Book management opens for Manager login

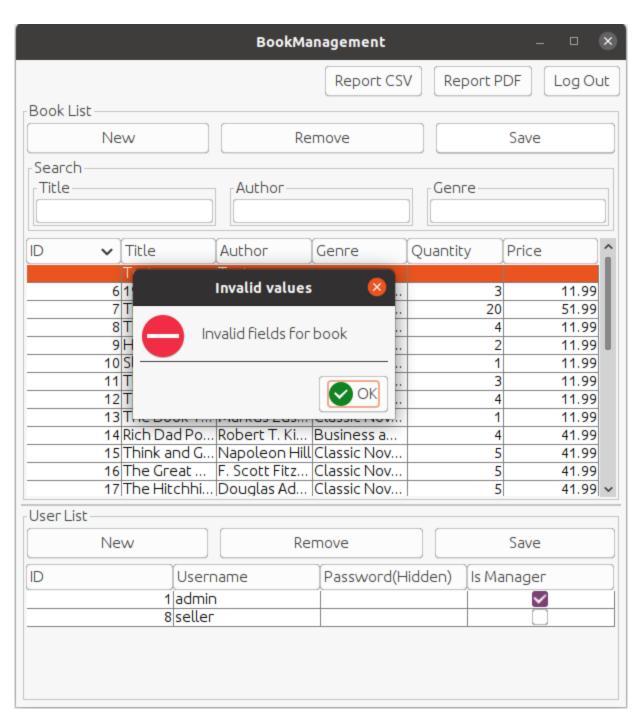


Figure 7 Try to add book with empty fields

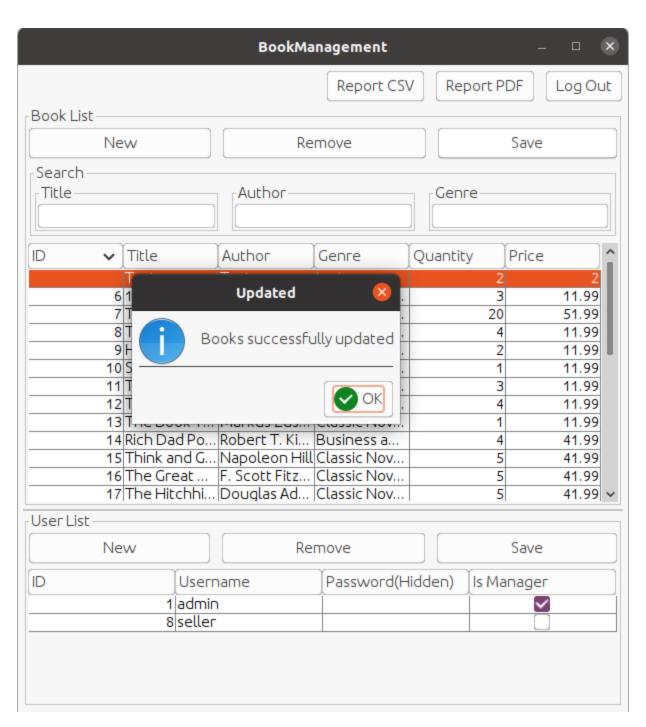


Figure 8 Add/modify book with correct fields

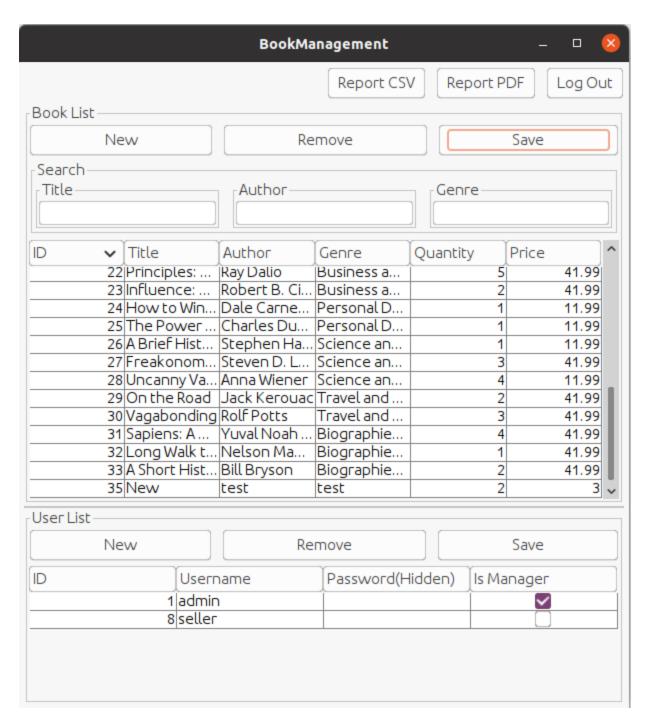


Figure 9 Save new book with incremental ID

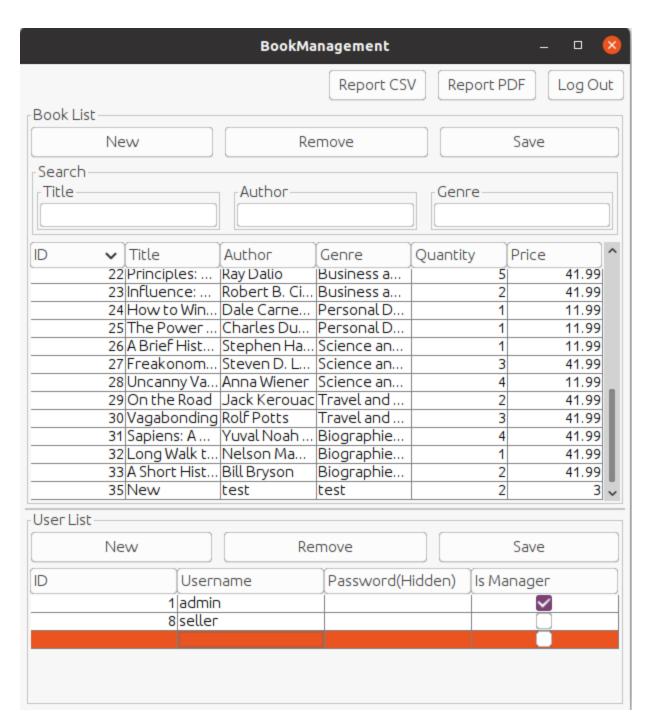


Figure 10 Try to add new user

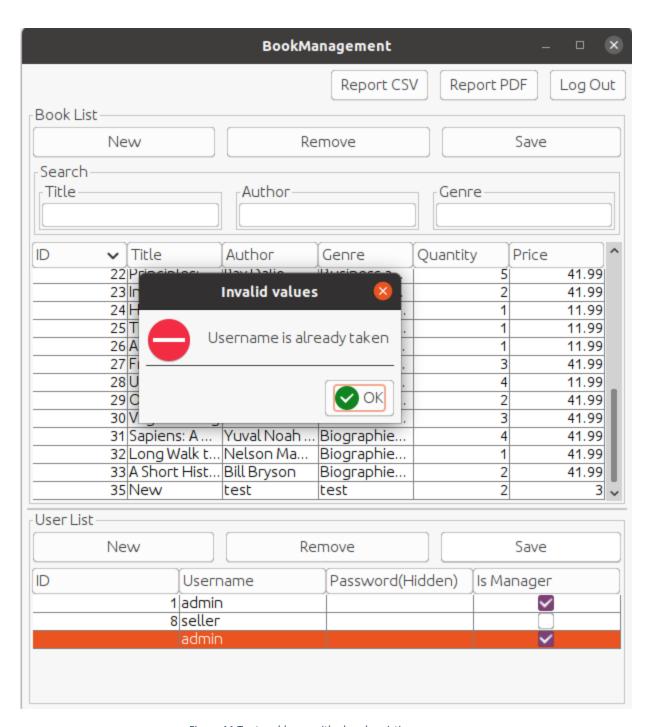


Figure 11 Try to add user with already existing username

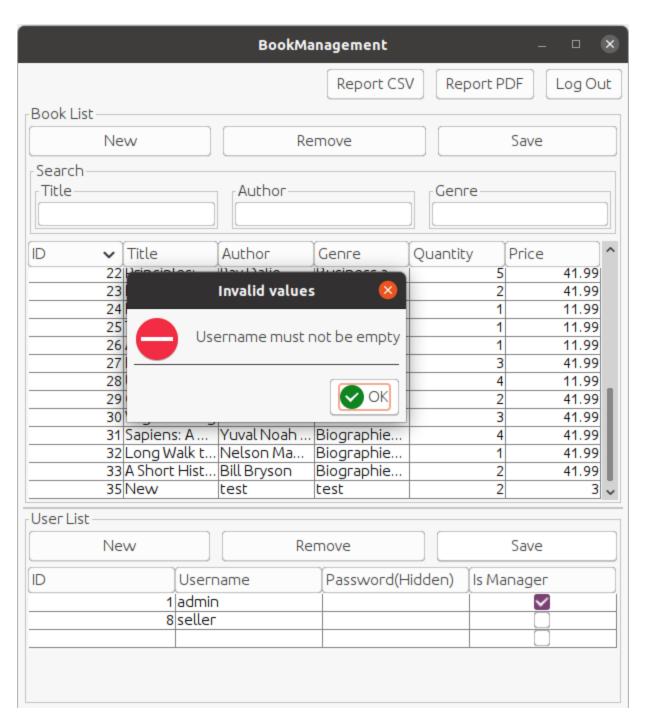


Figure 12 Try to add empty username

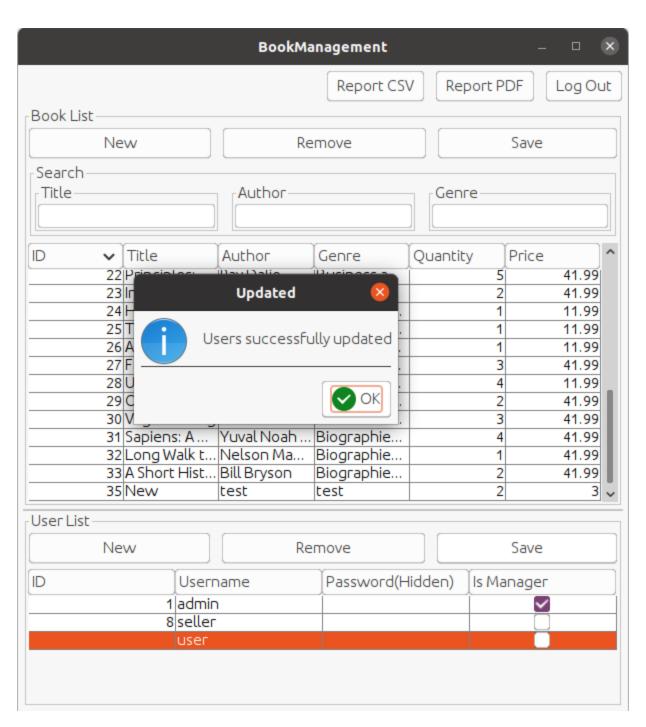


Figure 13 Add/save user information

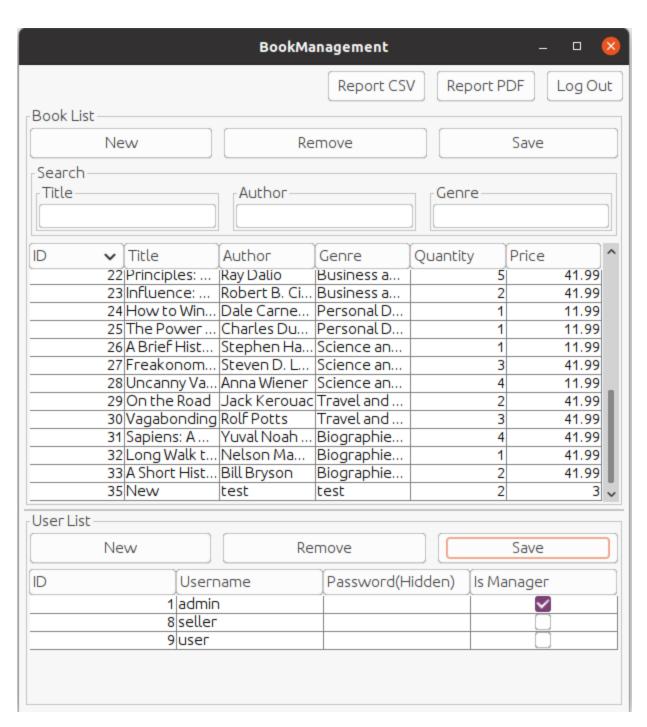


Figure 14 User added with incremental ID

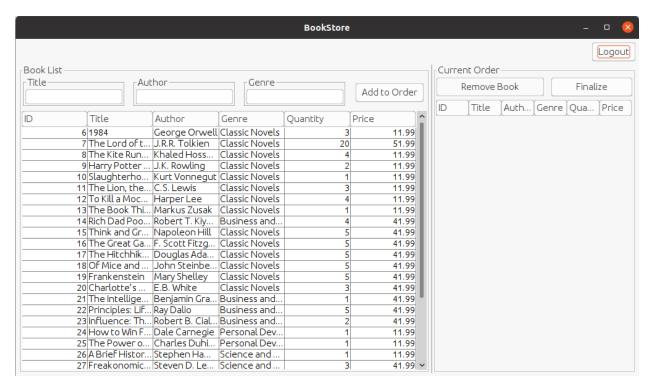


Figure 15 Book Store dashboard opens for seller user

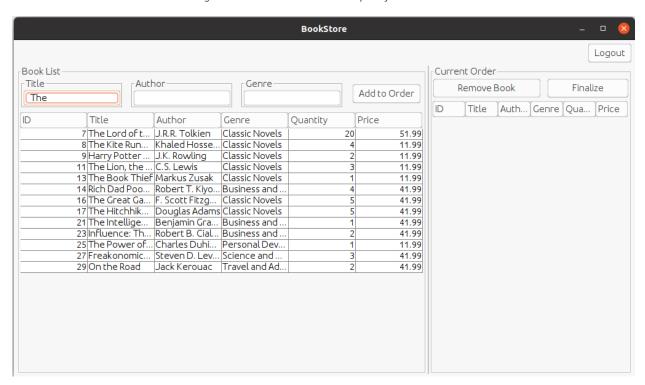


Figure 16 Single filter works

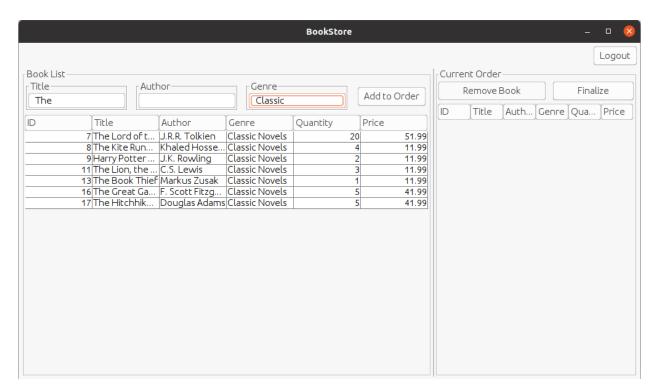


Figure 17 Multiple filter works

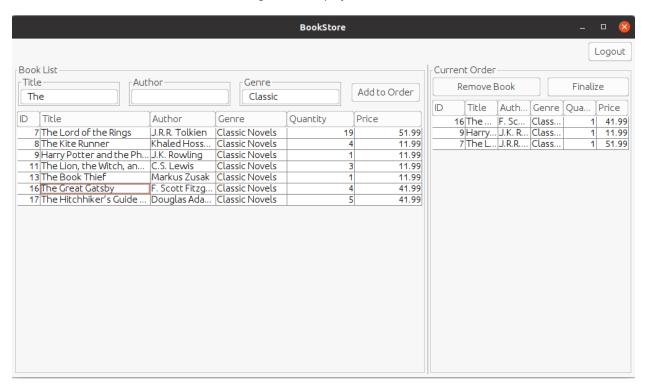


Figure 18 Add books to order works

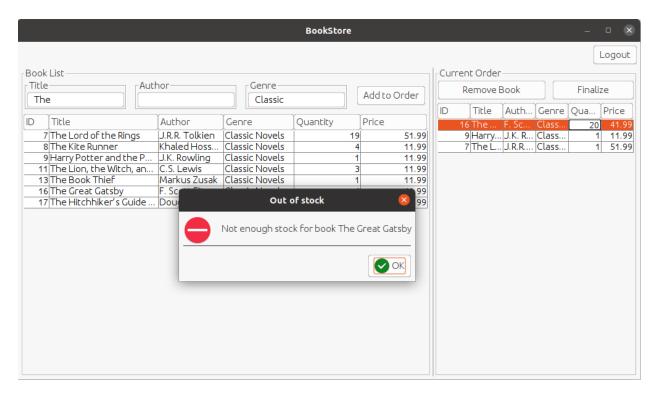


Figure 19 Try to modify stock for a book with not enough stock

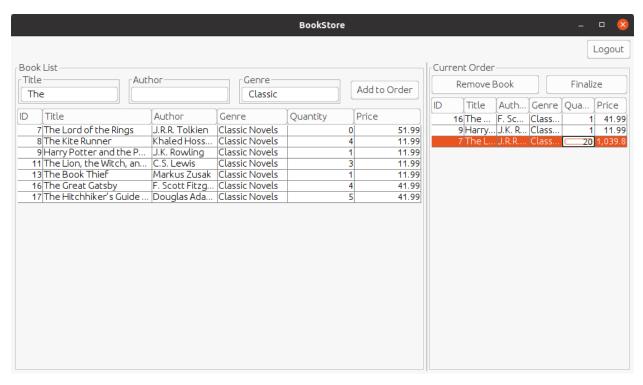


Figure 20 Try to modify stock for a book with enough stock

8 BIBLIOGRAPHY

MVC Design Pattern