Singapore University of Technology and Design  
Information Systems Technology and Design  
50.008 Database  
  
Online Bookstore Project 2014

Contents

[Group Members 1](#_Toc406431148)

[Key Folders 1](#_Toc406431149)

[ER Diagram 2](#_Toc406431150)

[Schema Design 3](#_Toc406431151)

[Implementing CHECK Constraints in MYSQL 5](#_Toc406431152)

[Initial Insertion of Test Data 6](#_Toc406431153)

[SQL Queries 7](#_Toc406431154)

[Implementation 10](#_Toc406431155)

[Screenshots of Application 11](#_Toc406431156)

## Group Members

1. William Haw (1000043)
2. Cheah Yew Hong, Kenneth (1000124)
3. Lee Ting’en Kevin (1000139)
4. Scott Chiang Chun Wei (1000238)
5. Immanuella Lim Hao Ni (1000342)

## Key Folders

|  |  |
| --- | --- |
| **Path** | **Description** |
| BunsAndNoodle\ | Parent directory |
| BunsAndNoodle\SQL files | Contains the SQL files for the schema, triggers, and insertion of initial data |
| BunsAndNoodle\bns | Contains the application developed using NetBeans IDE 8.0.2 running on Glassfish Server 4.1 and MYSQL 5.6.2 |
| BunsAndNoodle\bns\web | Contains all of the JSP pages which were coded |

Table 1: Key folders

## ER Diagram

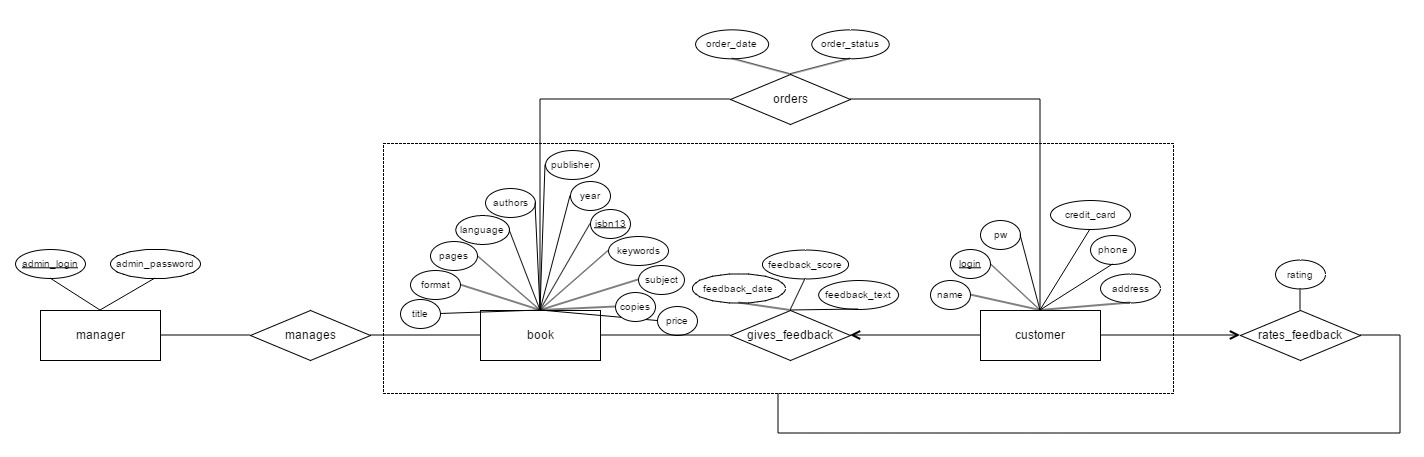


Figure 1: ER diagram

Based on the project requirements, we have designed the ER diagram above.

## Schema Design

We then proceeded to write the SQL statements for the tables based on the ER diagram, adding in additional CHECK constraints where it was applicable.

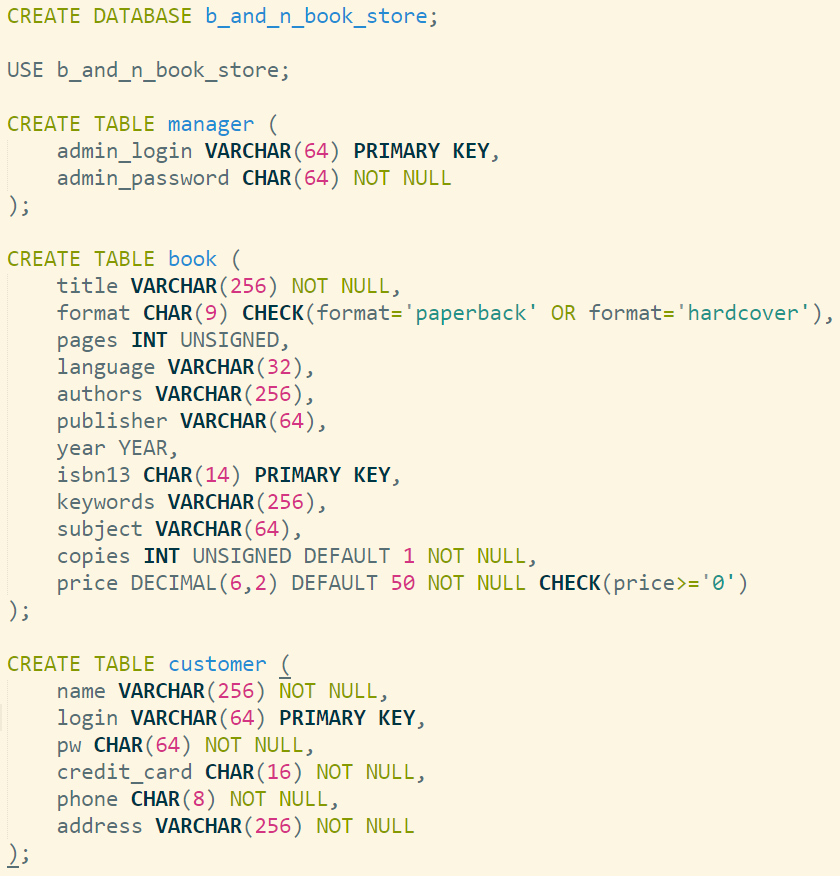


Figure 2: SQL statements for the manager, book and customer entities



Figure 3: SQL statements for the order, gives\_feedback and rates\_feedback relationships

## Implementing CHECK Constraints in MYSQL

Given that CHECK statements are parsed but not actually implemented in MYSQL, we created our own triggers to enforce the various CHECK constraints which we had created. For example, we needed to check that the book format is either ‘paperback’ or ‘hardcover’, or that the feedback score does not exceed 10, or that a customer cannot be rating his or her own feedback on a book etc. By enforcing such rules at the database level, it helped to ensure that the initial insertion of test data into the database conformed to those rules. It also minimized the likelihood of mistakes during application level development.

Here is a sample trigger which ensures the format of the book is either ‘paperback’ or ‘hardcover’ and that the book price cannot be negative.

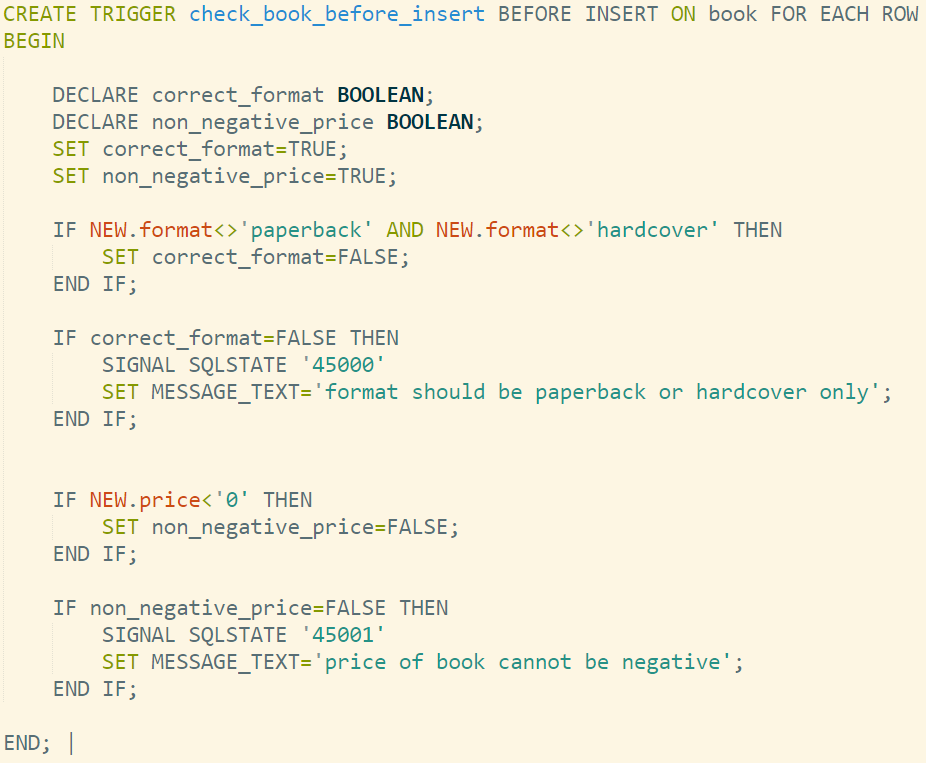


Figure 4: Sample Trigger

The rest of the triggers are located in BunsAndNoodle\SQL files\2.Triggers.sql

## 

## Initial Insertion of Test Data

As part of the development process, we had to create some dummy data for testing purposes. For example, after inserting the data, we tested out the various SQL queries to make sure they generated the correct output we wanted. Moreover, it allowed application developers to check if the results were being drawn out from the database and displayed on the webpage correctly.

In total, we inserted 100 legitimate books from 10 different subjects, 11 customers, 1 store manager, 3 orders, 10 book feedbacks (some feedbacks were applied to the same book by different customers so that we could calculate the average score) and 10 user ratings on the feedback given.

The SQL insertion statements for the various tables are located in BunsAndNoodle\SQL files\.

## SQL Queries

|  |  |
| --- | --- |
| **System Functionality** | **SQL Statements** |
| User registration | insert into customer (name, login, pw, credit\_card, phone, address)  values (fullname, user, pwd, credit\_card, phone, address) |
| User login | select \* from customer where login = userid and pw = pwd |
| User record – account information | select name, credit\_card, phone, address from customer  where customer.login = ? <sql:param value=“${userid}”/> |
| User record – history of orders | select title, order\_date, order\_copies, order\_status from orders inner join book on book.isbn13 = orders.order\_isbn13 where order\_customer = ? <sql:param value=“${userid}”/> |
| User record – history of feedbacks | select title, feedback\_date, feedback\_score, feedback\_text from gives\_feedback inner join book on book.isbn13 = gives\_feedback.feedback\_isbn13 where feedback\_customer = ? <sql:param value=“${userid}”/> |
| User record – history of usefulness ratings on feedbacks | select title, name, feedback\_text, feedback\_score, rating  from rates\_feedback inner join gives\_feedback  on (ratee, ratee\_feedback\_isbn13) = (feedback\_customer, feedback\_isbn13)  inner join customer on ratee = customer.login  inner join book on ratee\_feedback\_isbn13 = book.isbn13  where rater = ? <sql:param value=“${userid}”/> |
| User browses books by conjunctive queries, with sorting by year or average score of feedback | select title, format, pages, language, authors, publisher, year, isbn13, subject, copies, price, round(avg(gives\_feedback.feedback\_score), 2) as avg\_score from book left join gives\_feedback on book.isbn13 = gives\_feedback.feedback\_isbn13  where book.title rlike ? <sql:param value=“${param.title}”/>  and book.authors rlike ? <sql:param value=“${param.authors}”/>  and book.publisher rlike ? <sql:param value=“${param.publisher}”/>  and book.subject rlike ? <sql:param value=“${param.subject}”/>  group by book.isbn13  order by avg(gives\_feedback.feedback\_score) desc  // order by year desc; |
| For a given book, user views book details | select title, format, pages, language, authors, publisher, year, isbn13, subject, copies, price from book  where book.isbn13 = ? <sql:param value="${param.isbn13}"/> |

|  |  |
| --- | --- |
| For a given book, user views all the feedback and ratings on it (top n) | select feedback\_text, feedback\_score, feedback\_customer, name, feedback\_date, avg(ifnull(rating,0)) as avgScore, count(rating) as ratingCount from (select feedback\_customer, feedback\_date, feedback\_score, feedback\_text, name, feedback\_isbn13 from gives\_feedback  inner join customer on customer.login=gives\_feedback.feedback\_customer  where feedback\_isbn13 = ? <sql:param value="${param.isbn13}"/>) as K  left join rates\_feedback on rates\_feedback.ratee=feedback\_customer  and ratee\_feedback\_isbn13=feedback\_isbn13  group by feedback\_text, feedback\_score, feedback\_customer, name, feedback\_date  order by avgScore DESC  limit <c:out value="${toView}"/>; |
| For a given book, user enters in his or her feedback | "insert into gives\_feedback(feedback\_isbn13,feedback\_customer,feedback\_date,feedback\_score,feedback\_text) " + values ('" + isbn13 + "','" + user + "','" + currentTime + "','" + score + "','" + feedback + "')"); |
| For a given book, user rates the usefulness of other users’ feedback | "insert into rates\_feedback(rater,ratee,ratee\_feedback\_isbn13,rating) " +  "values ('" + user + "','" + ratee + "','" + isbn13 + "','" + score +"')"); |
| For a given book, user orders x copies of it (insert new order and update the book quantity) | "insert into orders(order\_date, order\_status, order\_isbn13, order\_copies, order\_customer) " + "values ('" + date + "','" + status + "','" + isbn + "','" + order\_copies + "','" + cust\_name + "')");  "update book set copies = '" + new\_copies + "'where isbn13='" + isbn + "'"); |
| User gets book recommendations after ordering books, sorted by most popular recommendations first | select title, format, pages, language, authors, publisher, year, isbn13, subject, copies, price from book  where isbn13 in (  select order\_isbn13 from orders  where order\_customer in (  select order\_customer from orders  where order\_isbn13 = "<%= request.getParameter("isbn13")%>"  having order\_customer <> ? <sql:param value="${userid}"/>)  group by order\_isbn13  having order\_isbn13 <> "<%= request.getParameter("isbn13")%>"  order by count(\*) desc);  Note ‘<>’ means ‘Not equals’ |

Table 2: SQL queries for users

|  |  |
| --- | --- |
| **System Functionality** | **SQL Statements** |
| Store manager – login | select \* from manager  where admin\_login = admin  and admin\_password = pwd |
| Store manager – enter new book | "insert into book(title, format, pages, language, authors, publisher, year, isbn13, keywords, subject, copies, price) " +  "values ('" + title + "','" + book\_format + "','" + Integer.parseInt(pages) + "','"  + language + "','" + authors + "','" + publisher + "','"  + year + "','" + isbn13 + "','" + keywords + "','"  + subject + "','" + Integer.parseInt(copies) + "','" + Float.parseFloat(price) + "')" |
| Store manager – update book copies | "select copies from book where isbn13 = '" + isbn13 + "'" current\_copies = rs.getInt("copies");  total\_copies = current\_copies + Integer.parseInt(number\_new\_copies);  "update book set copies = '" + total\_copies + "'where isbn13='" + isbn13 + "'" |
| Store manager – view top m popular books per month | select title, sum(order\_copies) as summed\_copies, order\_date, order\_isbn13  from orders inner join book on order\_isbn13 = book.isbn13  where month(order\_date) rlike ? <sql:param value="${param.month}"/>  and year(order\_date) rlike ? <sql:param value="${param.year}"/>  group by order\_isbn13  order by summed\_copies desc  limit <c:out value="${param.number\_results}"/> |
| Store manager – view top m popular authors | select authors, title, sum(order\_copies) as summed\_copies, order\_date, order\_isbn13 from orders inner join book on order\_isbn13 = book.isbn13  where month(order\_date) rlike ? <sql:param value="${param.month}"/>  and year(order\_date) rlike ? <sql:param value="${param.year}"/>  group by order\_isbn13  order by summed\_copies desc  limit <c:out value="${param.number\_results}"/> |
| Store manager – view top m popular publishers | select publisher, title, sum(order\_copies) as summed\_copies, order\_date, order\_isbn13 from orders inner join book on order\_isbn13 = book.isbn13  where month(order\_date) rlike ? <sql:param value="${param.month}"/>  and year(order\_date) rlike ? <sql:param value="${param.year}"/>  group by order\_isbn13  order by summed\_copies desc  limit <c:out value="${param.number\_results}"/> |

Table 3: SQL queries for store manager

## Implementation

We developed the application using NetBeans IDE 8.0.2. The backend server was running on Glassfish Server 4.1 which communicated with our MYSQL 5.6.2 database.

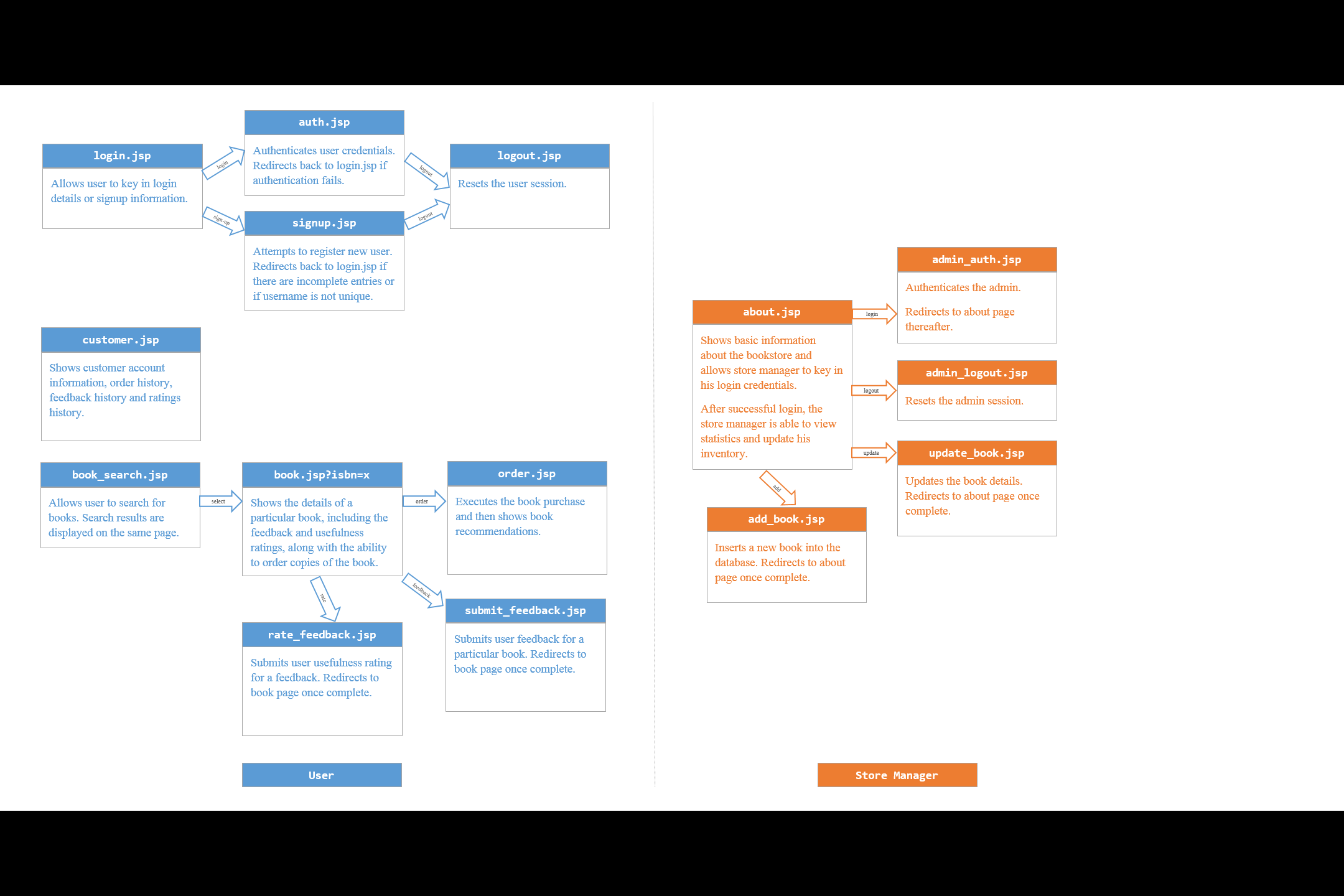
Here is a high level schematic of our various JSP pages and their key functions.

Figure 5: Schematic of JSP pages

## Screenshots of Application

Here are some screenshots of our web application.

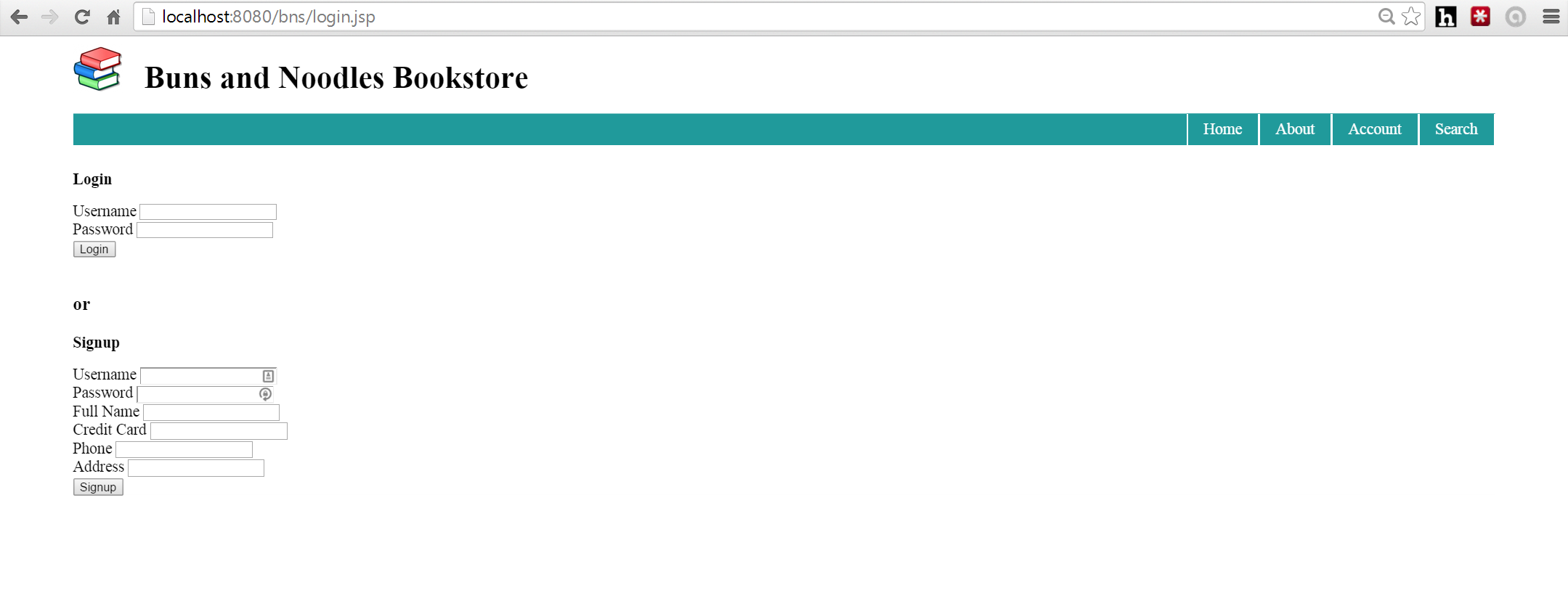


Figure 6: Login and sign-up page



Figure 7: Book search page with hyperlinks to individual book pages



Figure 8: Individual book page



Figure 9: Recommendations after purchase

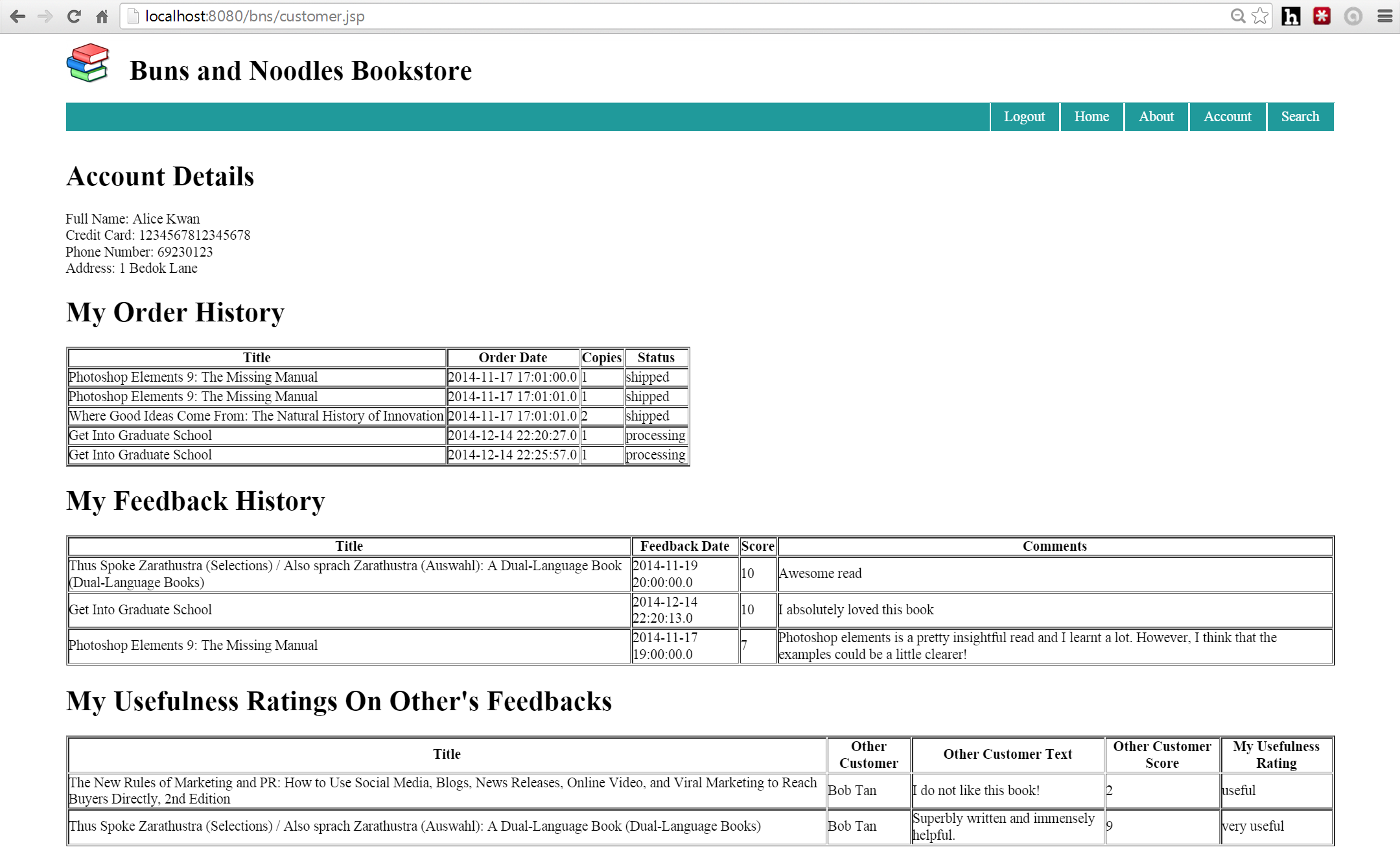


Figure 10: Customer account page

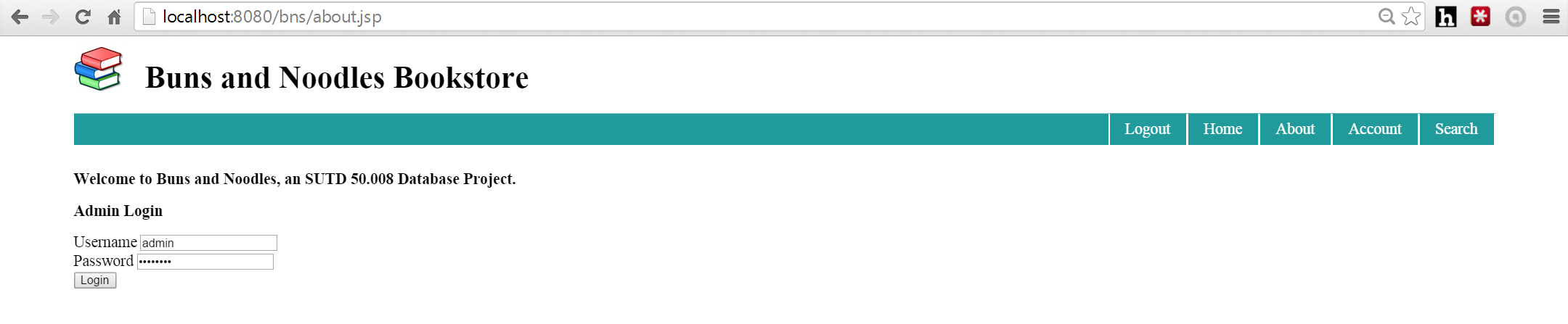


Figure 11: Store manager login

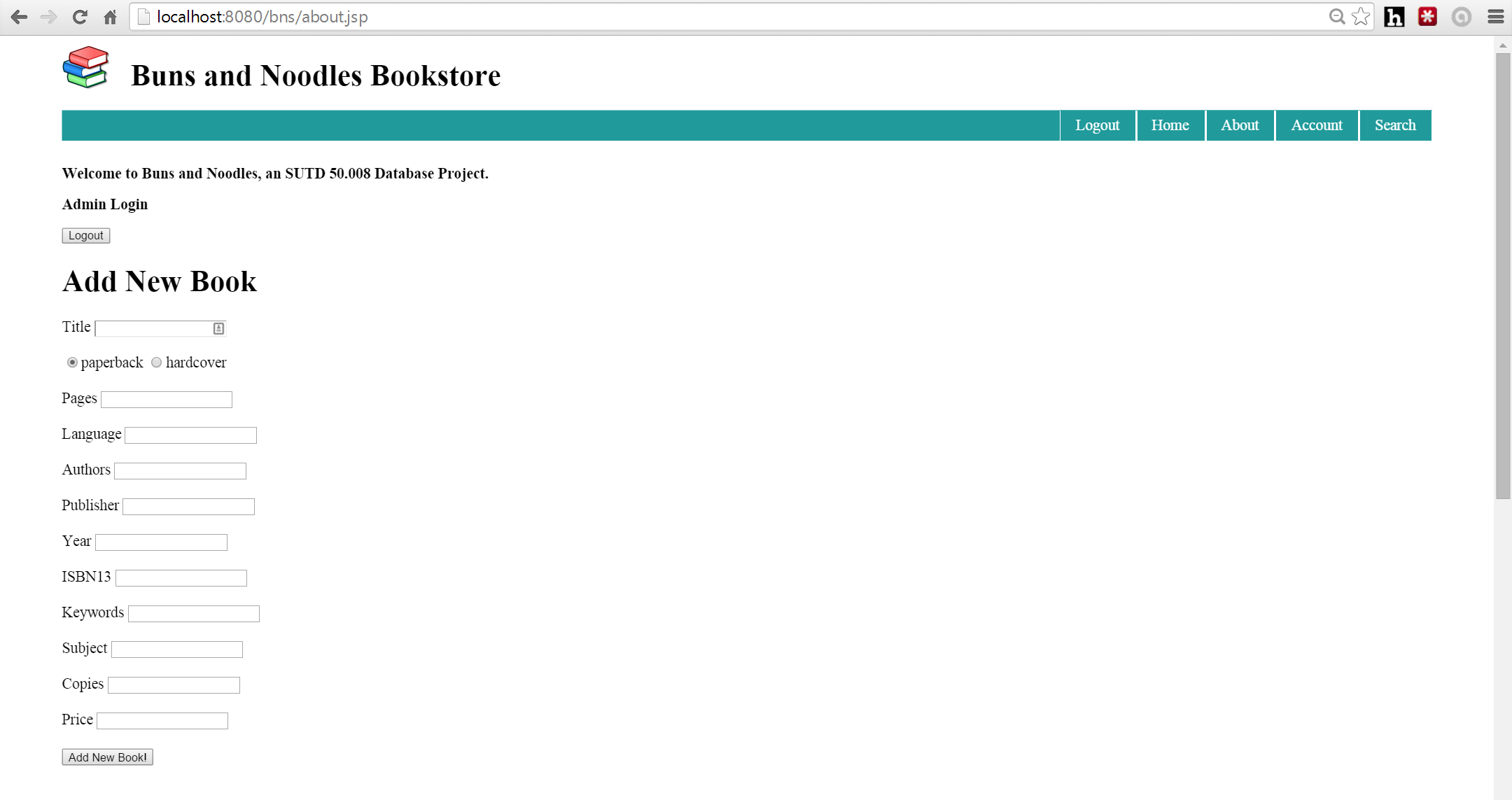


Figure 12: Store manager page (1)

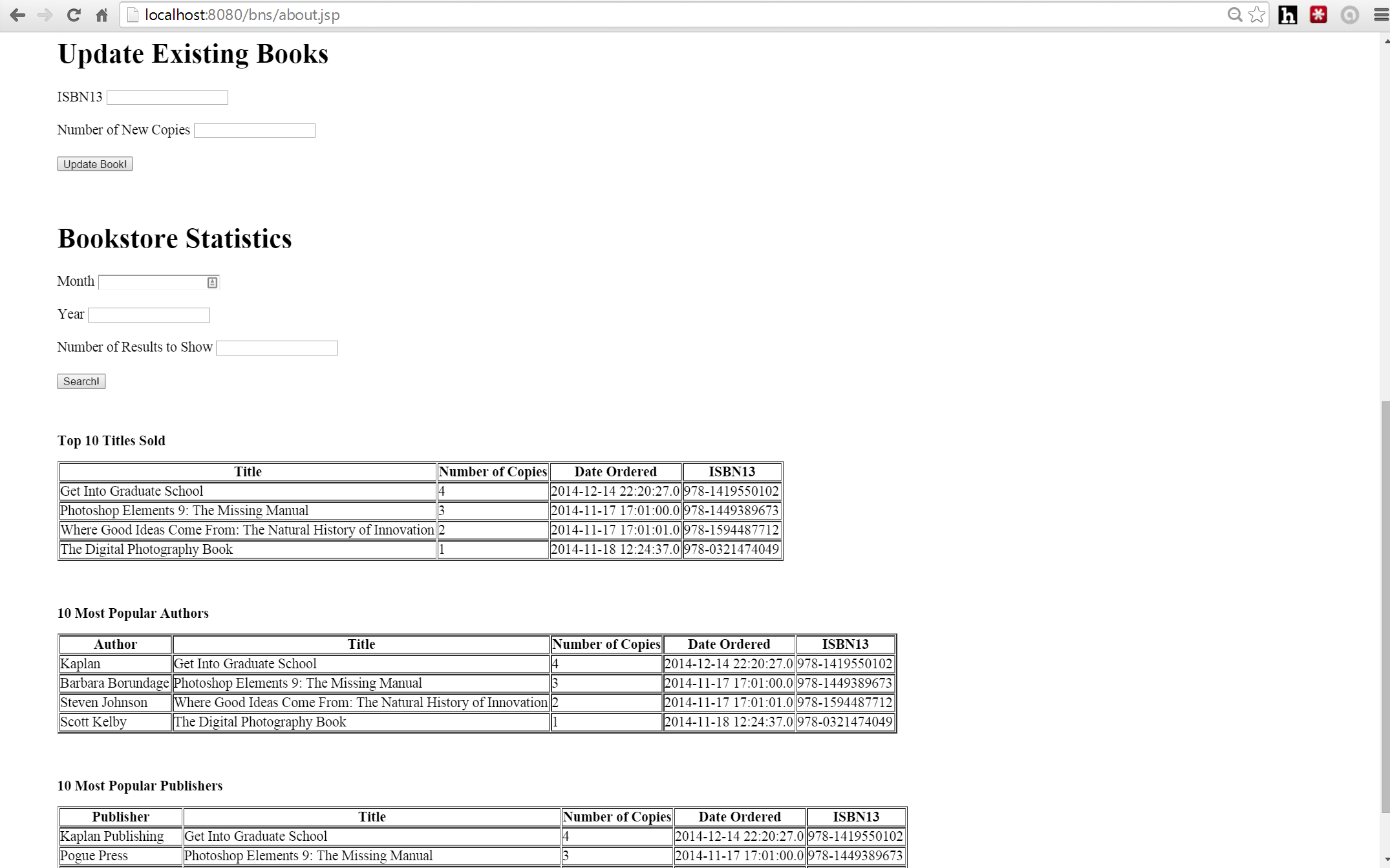


Figure 13: Store manager page (2)