Developing and implementing an E-Commerce system for an online bookshop

UP939702

Contents

Introduction

This paper aims to document the design and development of a functional dynamic web application making use of Java EE technologies. This will come in the form of an E-commerce system for the fictional cricket bookshop “Stumps”. Requirements for the program have been created and outlined in another document – ideally if these requirements are all met, the result will be an effective usable artifact. When accessing the site, users will be able to perform tasks such as searching for items, adding items to their cart, and placing orders. These examples of functionality along with many others when used in conjunction will need to be constructed in an efficient and useable manner.

Design

I decided to split my project up into different packages in order to separate and compartmentalise each of the different components i.e., the controllers and the business logic. There were many reasons for this but the main one being so that functions “can be can be written and encapsulated to become a reusable objects” (Bainbridge et al, 2001). For similar reasons, JSF pages were made for different main states of the website.

The controllers are split into 6 – one for each of the core pieces of functionality. These different controller beans will then interact with the corresponding service from the business logic package for example “AddUserBean” interacts with “AddUserService” which then constructs an instance of the entity “UserAccounts”. This new object is then passed into the corresponding façade so that it can interact with the correct table in the database.

The database was designed to split up the three main entities: Books, Users, and orders. Each of these had their own table in the database.

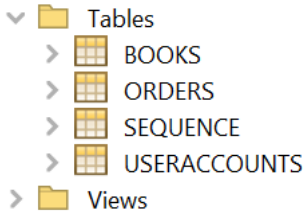

Figure 1 - Showing the tables within the database

This flow that has just been discussed is one that has been followed as it makes the flow of information "simple, efficient, and robust" (Prajapati & Dabhi, 2009) which allows for the production of a "high quality of web-application" (Prajapati & Dabhi, 2009).

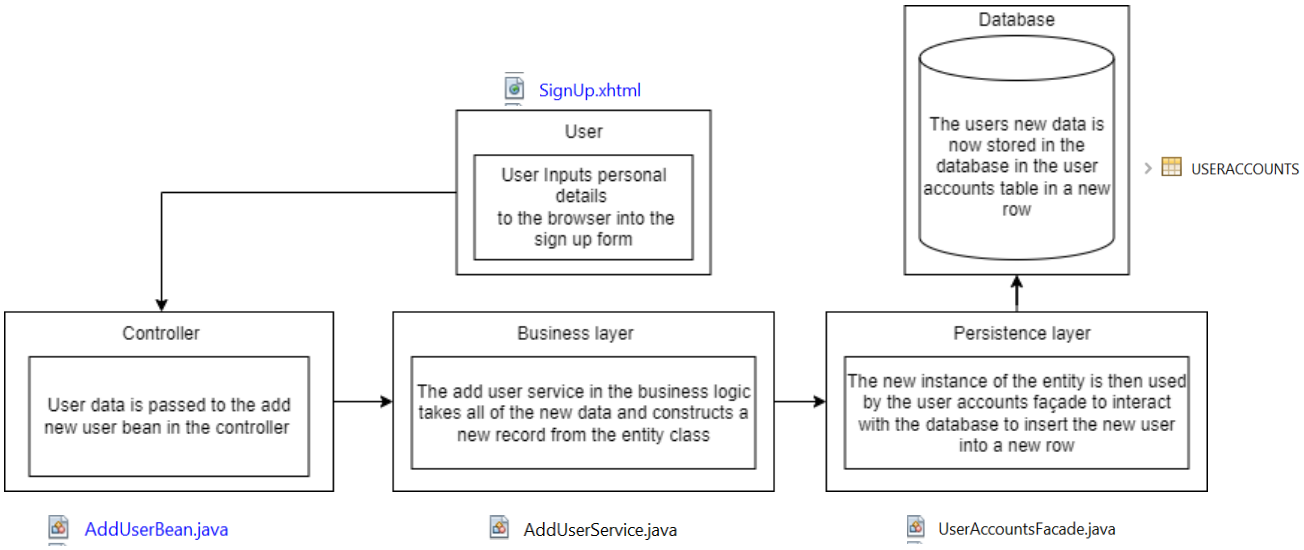

Figure 2 - Showing the flow of information when a user creates an account

Figure 2 is just one example of the many flows of information within the program. All the arrows are unidirectional as this flow is only storing a new user in the database – there is no data being returned.

Implementation

This section will discuss the development lifecycle of the project

Development tools

NetBeans

I chose to use NetBeans as my IDE as I had already had experience with it and somewhat knew my way around it. When downloading NetBeans, it comes bundled with other tools such as maven and glassfish. It also has the capability to provide templates of different file

types such as session beans or JSF pages – this allows for more efficient and effective use of time.

Glassfish

As mentioned, Glassfish comes bundled with NetBeans which is the primary reason as to why I decided to use it as my server. For the purposes that I needed the server for, the configuration that worked for me was version 5.1 of Glassfish and version 8 of Java. Another not so prominent reason for this choice is the error tracking, Glassfish's console output makes it very easy to understand and locate errors.

GitHub/GitKraken

I created a GitHub repository for the purpose of version control, this meant that when there was an error which I could not fix or if I deleted a file by mistake, I could revert to an older version where the code was in proper working order. It also meant that I was able to work from both my laptop and my PC as I could easily clone the most recent version of the repository.

I made use of GitKraken simply because it shows everything such as branches in a friendly and clear user interface rather than text output in the terminal.

Chrome

I used chrome as my browser of choice as it robust and reliable. Although I did not make extensive use of it, it should also be noted the quality of its console as it further aids error tracking.

Implementation

During the development process, as expected there were many obstacles and problems that had to be overcome/solved however some of them were more prominent that others.

When attempting to fetch a list of all the existing book entities from the database and then store them in a table for the user to see, initially within the query I was only selecting the information that I wanted to display such as the title of a book(b.title or b.author). It was only after lots of testing and research that I figured out that it would be a better idea to return the whole book object and then within the JSF page retrieve the relative attributes from the object. In the "AbstractFacade "class, the code that is automatically generated has a built in function "findAll()" which allowed me to return all of the books as objects so that they could then be manipulated and shown in the table as was needed.

One of the other issues that I encountered was once again regarding querying the database. As mentioned, there were some inbuilt functions within the "AbstractFacade "class however when the user searches for an item a custom query needed to be created and run. This proved somewhat difficult as not only did information need to be gathered from the user and then sent within the query to the database, but data also needed to be retrieved as well in the correct format and returned to the JSF page.

```
@Entity
@NamedQuery(name = "books.findByName",
            query = "SELECT b FROM books b WHERE b.title LIKE :name")
```

Figure 3 - Showing an example of one query

Once a user had logged successfully it was necessary to keep them logged in for obvious reasons. This was the most challenging problem that I faced during the whole development section of the project. Although I found a solution which I will discuss, I am aware that it is not the most effective or efficient and therefore would need further improvement in the future. Once the user logged in, I stored their email/ID in a private variable within the same controller named “activeUser” and then made a set of small functions that could interact and manipulate this value such as logging the user out. Although this bean was session scoped so the user would be logged in until they log out or the session ended, I found it challenging attempting to access these functions from within other beans. This proved a big issue when it came to the user trying to checkout as I was unable to identify who was logged in.

I tried to move this user information to a service class in the business logic but unfortunately, I was unable to successfully rectify this issue.

Once the user logged in, they were directed to the catalogue page which shows them all of the books available to buy. The navigation bar at the top consists of an assortment of buttons, if the user is a verified admin, then a button labelled Show all orders can be clicked however if they are not an admin then this button will not be visible.

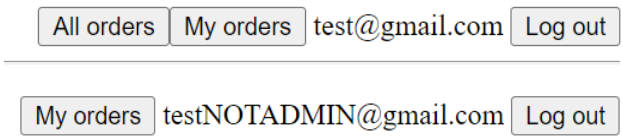

Figure 4 - Showing difference between admin states

To implement this functionality, I made use of the “render” attribute that can be applied to command buttons. For the value of the attribute, I referenced a function which queried the user database and returned the relevant Boolean value to see whether the user had admin status or not.

```
public String adminOrders() {
    if ((us.searchRecord(activeUser, password)).getAdmin() == true){
        return "true";
    }else{
        return "false";
    }
```

Figure 5 - Showing further admin control examples

The search book algorithm initially cased some difficulty as when I searched for a book that didn't exist the whole program crashed but after adding some exception handling in It was able to catch these errors. I initially tried to incorporate some Ajax so that when the user typed something into the search bar, the table would automatically get updated with the book names that contained whatever was currently in the search field. Unfortunately, this was something that I was not able to successfully implement and therefore to resolve this issue I decided to add in a standard search button that executes the query in the database and then updates the results table.

Another issue regarding the search function was that the add to basket buttons next to each of the table rows were/are very temperamental and unpredictable. It seems that there is no pattern as to why they work or not and therefore to guarantee the users book being added to the cart, the user must do this through the catalogue page.

Testing

Testing was continuously conducted in a modular fashion – after any change was made, it was tested and if the change did not perform as desired further changes would not be made until the desired functionality was reached. This method is based on the incremental methodology which is one that I had worked with before and was suited to this type of project.

As well as testing each increment as a change was made, I also periodically tested the whole system in its current state to see whether all the components were working together as a whole. As this project was not exceedingly large, I deemed it unnecessary to apply/make use of a proper testing framework such as selenium and therefore stayed with manual testing.

Summary

During this coursework I designed and created a functioning E-commerce book shop using Java EE technologies. I applied both theoretical knowledge and practical knowledge that I had recently learned to help implement many different key features to the site to provide the core functionality that would be needed for real world deployment. Although for most of the requirements I was successful, there were some that I was unable to fulfil.

As mentioned earlier in this paper, the method that I used to store the active user in the session was not the most effective and therefore an alternative approach such as storing the user info in within the business logic or in an "HTTPSession" would most likely have proven more efficient.

I structured the application in such a way that it was easy and logical to navigate, it also proved very robust and all the interactions such as those with the database worked as expected.

References

Bainbridge, A., Colgrave, J., Colyer, A., & Normington, G. (2001). CICS and enterprise JavaBeans. *IBM Systems Journal*, *40*(1), 46-67.

Prajapati, H. B., & Dabhi, V. K. (2009, March). High quality web-application development on java EE platform. In *2009 IEEE International Advance Computing Conference* (pp. 1664-1669). IEEE.

What Is Incremental Testing: Detailed Explanation With Examples | Software Testing Help | (n.d.). | May 2, 2022, from | https://www.softwaretestinghelp.com/incremental-testing/

Java Platform, Enterprise Edition: The Java EE Tutorial | Oracle (n.d.). | May 7, 2022 from| https://docs.oracle.com/javaee/7/tutorial/ejb-basicexamples001.htm

@Stateful vs. @SessionScoped | Adam-Bien | Adam Bien | May 10, 2022 from | https://www.adam-bien.com/roller/abien/entry/stateful_vs_sessionscoped

When should we implement Serializable interface? | Stack Overflow | May 9, 2022 | from | https://stackoverflow.com/questions/4548816/when-should-we-implement-serializable-interface