**HITTASTIC WEB APPLICATION USING JAVA JSP AND MYSQL**

**COM 528(OOP)**

**ASSESSMENT REPORT**

BY

DAVID AGBOLADE

STUDENT NO.: Q14872382

BSC(Hons) SOFTWARE ENGINEERING

**INTRODUCTION**

Users of Hittastic can browse through their orders and view the history of their orders as well as search for and purchase any kind of music they prefer. Users can also add goods to their baskets. MySQL was used as the database for this project, which was created in Java JSP. Various user levels will be available on the system, including admin and customer/user.

**SYSTEM REQUIREMENTS**

* Java JDK version 8 and above.
* Apache tomcat.
* Xampp
* MySQL for the database.
* NetBeans.

**IMPLEMENTATION**

Therefore, I began by cloning the group's basic solution from GitHub and executing it on NetBeans, after which I built a new web project and followed the notes to acquire the idea since I did not complete the group project. I uploaded the user, order, and song class files to the project and modified them by adding user-id, quantity, and date to the order class, adding a genre, making the amount a double in the song class, and changing the user class to email instead of having users log in with usernames, also created a class file for the cart and admin login.

Text, table

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

For a web application, the data access layer is responsible for interacting with the database and performing CRUD operations. This layer is often implemented using Data Access Objects (DAOs), which provide a standardized interface for interacting with the database.

The rendering of the user interface, management of HTTP requests and answers, and user interaction all fall under the purview of the presentation layer. Servlets, which are Java classes that manage HTTP requests and responses, are frequently used to implement this layer.

The application logic layer takes care of constructing the essential features of the programme, including input validation, and communication with the data access layer. Model classes, which represent the data and behaviour of the application, are frequently used to implement this layer.

Whilst few of the JSP files have their Dao in them, It is simpler to understand and maintain the code by separating these folders in the source package.

**TASKS**

1. A user should be able to search for songs by artist and by title:

I had a bit of a struggle when implanting this task, as the search results were not giving the expected result as t could search for either only the title or only the artist, at some point, I thought of creating 2 search bars one for the artist and one for the title but realised it won’t be a clean code even though it works. I ended up fixing it at the latter by getting the value of the parameter and saving it in a string variable.

Text

Description automatically generated

The try-catch block's code creates a connection to a MySQL database, prepares a SQL statement that searches for rows in the songs table where the search input is present in the artist or title field, and then runs the statement.

While iterating through the query's results, the while loop creates a new Song object for each row and sets the properties of that object to the values of the columns in that row.

The song list is then updated to include the Song object.

The request field songs store the list of songs, which is accessible to other application components like a JSP page that shows search results.

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

The above result was a search for an artist.

Graphical user interface, text, application, chat or text message

Description automatically generated

The above result was a search for the title only, so users can search for both artist and title.

1. A) A user should be able to buy one or more copies of a song:

For this task, I created 2 buttons using that is going to increase or decrease the number of songs that the user wants to buy.

Graphical user interface, application

Description automatically generated

Then I created a servlet that would serve as a link to increase or decrease the number of songs.

Graphical user interface, text, application, email

Description automatically generated

The codes above show The Print Writer object named out is created by the try block using a try-with-resources statement and used to write a response to the client.

The action and id arguments are taken from the request and put into local variables by the code. The id parameter specifies the ID of the item in the cart, and the action parameter defines whether to increase or decrease the quantity of the item.

The session's list of Cart objects, which represents the items in the cart, is where the code receives the cart-list attribute.

B) users should be able to add songs to the cart: for this task, I created a servlet that would serve as a link to the add to cart button.

Graphical user interface, text, application

Description automatically generated

The code initialises a Cart object called an item and generates an empty list of Cart objects called cart-List. The quantity field of the item object is set to 1, and the id field is set to the id argument from the request.

The function retrieves the cart-list attribute from the session, which is intended to be an Array List of Cart objects and obtains the current session for the request. The code sends the client back “home. JSP” page if the cart-list attribute is nil and sets the cart-List attribute to the session.

The code sets the cart-List to the cart-list attribute and initialises the existing Boolean variable to false if the cart-list attribute is not null.

Once the user buys a song or checkout, an order is then created and saved on the orders page, and then the songs bought will be removed from the cart page. I had a few challenges when implementing the virtual bank balance and the quantity reduction, tried displaying it on the cart page but it keeps giving some errors, so I ended up removing it.

Graphical user interface

Description automatically generated

From the results above, users could add items to the cart and cancel a song if the user wants to. The total amount of all songs added to the cart is also added and when the user clicks on buy or checkout the songs would be added to the orders page.

1. A user should be able to view order history as a list.

I created a DAO to select all orders from the order table in the database column and joined it with the songs table.

Graphical user interface, text

Description automatically generated

From the image above, the list function generates a new empty List of Order objects. The method enters a try block and constructs a SQL query to select all rows from the orders table where the user id field coincides with the id parameter of the method. The statement is then carried out, and the outcomes are kept in an object called rs in a ResultSet.

After starting a while loop, the procedure loops through the rows of the ResultSet. The procedure initialises a SongsDao object called music and produces a new Order object called history for every row. A data access object called SongsDao offers a standardised interface for interacting with the database's songs table.

The getSingleSong function of the SongsDao object is used to retrieve a single Song object from the songs table after retrieving the song id column from the current row of the ResultSet. The procedure then assigns the values of the respective columns in the ResultSet and Song objects to the attributes of the history object.

The procedure moves on to the next row of the ResultSet after adding the history object to the list. The method returns a list of Order objects after the while loop.

Graphical user interface, table

Description automatically generated with medium confidence

From the results above, user can view all orders that has been bought and can also decide to remove the order from the list.

1. A user should be able to log in and the system must be able to distinguish between regular users and admin using session.

I created a servlet for this task and used a session-based login system.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

After receiving the login request, the servlet tries to verify the user's identity using a MySQL database. If the login is successful, a new HTTP session is created for the user, who is then forwarded to the proper page based on their role or status (either "home.jsp" or "adminIndex.jsp"). If the login is unsuccessful, it sends the request and answer to the "login.jsp" page and sets an error message as an attribute of the request object.

1. An Admin should be able to add a song:

Created a DAO for the insert method to insert songs into the database.

A picture containing line chart

Description automatically generated

A request to add a new song to a MySQL database is handled by what appears to be a Java server page in the code above. After receiving user input for the song title, artist, quantity, and genre, the server page utilises JDBC to establish a connection to the MySQL database and inserts a new entry with these values into the "songs" table. It prints a message stating that the song has been inserted if the insertion is successful. If an exception occurs, the stack trace is printed.

Graphical user interface, application

Description automatically generated

1. An admin user should be able to view a list of all users:

Created a Dao to select all users from the users table in the database and display them on the view-User page.

Graphical user interface, text, application

Description automatically generated

According to the code above, the Java class function receives a list of users from a MySQL database and returns it as a list of user objects. Using a JDBC Prepared Statement, the method establishes a connection to the database, runs a SELECT query to retrieve all rows from the user table, iterates over the ResultSet, generating a new user object for each result, and adds it to the list. User ID, email, and name fields in the user object are set using the matching values from the ResultSet. If an exception occurs, the stack trace is printed.

Graphical user interface, text, application

Description automatically generated

1. An admin user should be able to edit the details of a current user. I created an update query in the update file to update the user details and save it in the database.

Text

Description automatically generated

The code above creates a JSP page that modifies a row in the "user" MySQL database table. When a form submission is received, the server page utilises JDBC to establish a connection to the database, generate a prepared statement with an UPDATE query, and carry out the update. For the row with the given ID, the update changes the values of the "id," "name," and "email" columns to the new ones that are given. It prints a message stating that the record has been changed if the update is successful. If an exception occurs, it prints the exception and sets an error attribute. Graphical user interface

Description automatically generated

Graphical user interface, website

Description automatically generated

The results above show that when the admin clicks on the edit button, it goes to the update. Jsp page and display the form which will then update any user the admin selects.

B) An admin should be able to remove a user from the system:

Same as the update but this time using a delete query in the admin Dao.

Text

Description automatically generated

The code above uses a Java method to delete a row from the "user" table in a MySQL database. The method takes an integer ID as an argument, utilises JDBC to generate a prepared statement, and sets the ID as a parameter for the DELETE query. The delete statement is subsequently carried out by the procedure. If an exception occurs, the stack trace is printed.

Graphical user interface, table

Description automatically generated with medium confidence

When the user clicks on the remove button, the user gets deleted from the table.

**TESTING**

To consume the web API, I used Rester to test the search API and the view orders API.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

And build a test client to test the web API in java using restful API services and Scanner.util to perform some test clients

Graphical user interface, text, application

Description automatically generated

The above code is for a test client that utilises the Java API for RESTful Web Services to access the web API (JAX-RS).

With the query parameter "q" set to "Davido," the test client sends a GET request to the "http://localhost:8080/hitTasticc/search-results.jsp" URL. The request specifies that an answer with the content type application/json is acceptable.

The response status code and response body are then printed by the test client.

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Graphical user interface, text, application, email

Description automatically generated

I also used Junit and Mockito to test some of the files which are non-Gui-classes. In addition to other testing frameworks like JUnit and TestNG, Mockito is a java-based mocking framework. It enables the creation of service objects and internally uses the Java Reflection API. A mock object avoids external dependencies and returns fictitious data. Applying the mocks to the code being tested, makes the process of creating tests simpler.

Graphical user interface, text, application

Description automatically generated

The insertOrder method of the OrderDao class is tested in this test case's code.

The Connection and PreparedStatement objects that are utilised in the insertOrder method are mocked by the test case using mock objects. When specific methods are invoked on the mock objects, they are programmed to return predetermined values.

The test case then constructs a sample Order object, passes it as a parameter to the insertOrder method, and runs the method. The test case then confirms that the fake Connection and Prepared Statement objects had their methods called as expected and determines whether the insertOrder method provided the right result.

**TEST PLAN**

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Expected result | Actual Results | Reason |
| 1 | The user runs the application successfully | Build and run successfully |  |
| 2 | The User Logs in | The user logged in successfully |  |
| 3 | Add songs to the cart | Songs were added to the cart |  |
| 4 | Search for the artist and title | The user was able to search for both successfully |  |
| 5 | User Checkout | User able to checkout |  |
| 6 | The user is logged out of the account | Users can log out |  |
| 7 | The song is removed from the cart | Users able to remove songs from the cart |  |
| 8 | View orders on the orders page | Users can view orders history |  |
| 9 | Buy songs directly from the homepage | Users can buy songs from the homepage directly |  |
| 10 | Quantity should be increased or reduced | Users can increase or reduce the quantity |  |

Admin Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Expected Result | Actual Result | Reason |
| 1 | Admin logs in | Admin can log in |  |
| 2 | Admin can update user details | Admin can edit user details |  |
| 3 | Admin can remove user | Admin can delete User |  |
| 4 | Admin can view all users | Admin can view all users |  |
| 5 | Admin can add new songs | Admin able to add songs successfully |  |
| 6 | Admin can log out | error | An exception occurred processing |
|  |  |  |  |

How to run is included in the readme file

GitHub link: https://github.com/dagbolade/hitTasticc

CONCLUSION

In conclusion, I was able to learn a lot while creating this project, with the help of my little knowledge in web design, I added bootstrap to use as a basic styling for the front end. I added some authentication on the users’ page, where the system would make sure the user logs in again to either view orders or buy songs or checkout, same with the admin, the admin must log in again before performing any operations.

I came across many challenges when doing this project, some errors but able to fix most of them. I was debating on using an inner join for the SQL query to join the songs and orders table which would be better and faster instead of calling the getSingleSong in the SongsDao, but I had some errors while using inner join, would leave that for future Implementations, as I intend to add a payment gateway, add some images to the projects later, designs and implement the virtual balance and reduction of quantity when the user checks out.

References

* NOVI, P., 2019. *Step by Step: Making a Simple CRUD Application Using Java Servlet/JSP | Mitrais Blog* [online] [viewed 19 Nov 2022]. Available from: https://www.mitrais.com/news-updates/step-by-step-making-a-simple-crud-application-using-java-servlet-jsp/
* Pankaj, P.K. (2022) *Mockito tutorial*, *DigitalOcean*. DigitalOcean. Available at: https://www.digitalocean.com/community/tutorials/mockito-tutorial (Accessed: December 29, 2022).
* PFAFF, F., 2012. *Unit Tests with Mockito - Tutorial* [online] [viewed 31 Dec 2022] Available from: https://www.vogella.com/tutorials/Mockito/article.html
* SMIRNOV, A., 2020. *How to Create a Simple REST Client for Testing Your API.* [online] [viewed 31 Dec 2022]. Available from: https://itnext.io/how-to-create-a-simple-rest-client-for-testing-your-api-a0554d8380f8