Programming Project
Bookstore Web Application (Part 3)
Due: Thursday, April 21, 11.59PM (Blackboard)

Rigel Gjomemo

April 15, 2016

1 Description

The purpose of this project is to make you familiar with the way in which web applications are written and supported by a backend database. In this project, you will need to create a web application to implement a basic online book store. The web application will allow registered users to browse through the books and their descriptions, place orders, add reviews, and so on.

The project will be divided into several parts of increasing complexity. This document describes the second part, which is dedicated to adding more functionality to the web application.

Important Note: Since this is a database class, you are not expected to create a fancy client side of the application. However, you will need to become familiar with some basic HTML (e.g., forms). You will not be graded on how good the web site looks, but on the correct implementation of the functionality described in the rest of this document.

2 Setup

We will use the Apache Tomcat web server [1] and JSP [2] for this first project. JSP is a Java-based server side language, similar to PHP and ASP.net. In JSP, one writes Java code to process input from the user and send responses. The Java code can be mixed with HTML content and is executed on the server.

[1] https://tomcat.apache.org/download-70.cgi

[2] http://www.tutorialspoint.com/jsp/

2.1 Setup

To make sure there are no glitches during the submission, you will work on Eclipse and submit the Eclipse project. Create a dynamic web project on Eclipse and put the files from first project inside the WebContent directory of the project (this is the default name). That is, everything that was inside the `bookstore` directory in the first project goes in the folder WebContent in your Eclipse project. When running your project on Eclipse, you can specify an existing Tomcat installation or download and create a new Tomcat server.

3 Project Description

This section describes the high level functionality that you need to add to your web application and the different entities and how they interact with one another. You are free to come up with any design for the database that you deem useful.

3.1 Functionality

In the following paragraphs, the word MUST is in front of the required functionality. You can add more functionality and pages (which you should describe in your writeup, see Deliverables section below). This is not the only possible structure of the web application, however, and it is open for discussion. You may be able to change it if you want, provided the functionality does not change, and only after a discussion with me and/or the TA. In any case, you must provide a clear description of the workflow of your application in your writeup. The web application must check for several types of errors and conditions, described below:

1. **New username (10 points)** When the user signs up, the username must not exist. Therefore, before sending the insert query to the database, you must send a select query to see if the username exists in the users table. If the username exists, redirect the user to an error page with a simple error message and a link back to the Signup page.
2. **'Hard' Password (10 points)** The password that a user chooses when the user signs up or when (s)he (or the admin) changes it must be at least 8 characters long and be composed of letters and numbers. If the password does not match these characteristics the user must be redirected to an error page with a simple error message and a link back to the Signup page.
3. **Protection against SQL injection (20 points)**. All your queries must be transformed into prepared statements (10 points) in order to avoid SQL injection. Before doing this, in addition, all the strings containing input from a user, which go into a query must be sanitized. This involves giving the String to a function and getting back a String where single quotes and other characters are escaped, and so on. There are several libraries that do this. Check out for example `StringEscapeUtils` (`https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringEscapeUtils.html`). An

example of using this class (to create random user_id strings) is in the `Signup.jsp` file in my code of the bookstore application.

4. **Workflow (30 points)** If a user is not logged in, they cannot place orders. However, they can see the list of books. Therefore, you need to add code that, when a user tries to perform any functions (like order a book or change their information) without being logged in, (s)he is sent to the Login.jsp page instead. After logging in, the user should automatically be redirected to the page they tried to access. For instance, if they try to buy a book without being logged in they should be redirected to the Login page and after logging in, they should be redirected to the ShowBooks page. If they try to modify their information without being logged in, they should be redirected to the Login page and after logging in, they should be redirected to the page that modifies their information.

   This can be done in different ways. One solution would be to get the `referer` header, using the `request` object. **request.getHeader("referer")** returns the page from which the request is made. For instance, say the user is not logged in and is currently in the ShowBooks.jsp page. Say that the user clicks on the submit button to place an order for a book and the action in the correspoding form points to `InsertOrder.jsp`. If you call `request.getHeader(''referer'')` inside InsertOrder.jsp, it will return the URL of `ShowBooks.jsp` because `ShowBooks.jsp` is the page from which the request was made.

   Using this mechanism, you can get the referrer URL (ShowBooks.jsp), and when redirecting the user to Login.jsp, you can append the value to the URL of `Login.jsp` as a parameter. That is, instead of redirecting to just `Login.jsp`, you can redirect to `Login.jsp?returnPage=ShowBooks.jsp`. Inside `Login.jsp` you can retrieve the value of the parameter `returnPage` and use it to redirect the user back to that page.

5. **Hashed Passwords (15 points)**. So far, the password has been stored in clear (plain) text in the database. This is a security risk in case the database content is leaked. To address this risk, instead of storing the clear (plain) text passwords in the database, you must store the hash of the password in the password field, during Signup or when a password is modified. Then, when checking if a user entered the correct password you must compare the hash value of the entered password (in `Login.jsp`) with the value that is stored in the table.

   There are different ways to do this. See an example of how to get a hash value from a String here:
   `http://www.codejava.net/coding/how-to-calculate-md5-and-sha-hash-values-in-java`.
   Suggestion: you can put the whole code in a function that takes a String in input and returns the hash value in output and place that function in a jsp file, which you can include at the beginning of your JSP file (my code of bookstore has one such file, called CommonStuff.jsp).
   [3]

[3] You can use MD5 for the project but it is considered weak, that is given the hash it is possible to get the clear-text value relatively easily. The best hashing schemes to use are the so called slow hashes, where, if an attacker gets the hash, it will cost a lot to perform a brute force attack.

4 Deliverables

1. **Code**. Submit in a zip file (not tar.gz) the Eclipse folder of your project. Before compressing the Eclipse folder, you must dump your database to a SQL dump file named `bookstore.sql`. Then place this file inside the Eclipse folder. Then create a zip file of your Eclipse folder and submit it on Blackboard. Mysqldump instructions can be found at `https://dev.mysql.com/doc/refman/5.7/en/mysqldump-sql-format.html`.

2. **Writeup**. You must include a writeup that gives a detailed description of your application. In particular, it must contain the workflow (that is what each page does, and how the pages are connected to each other). The writeup must also contain a description of how we can actually use the web application, so that we can grade it correctly. Remember, we do not know the usernames and passwords that you are using, so include at least one in the writeup.

   The writeup must also contain feedback to us about the project. What level of difficulty it was, what part did you find the hardest to do, what was the most interesting part, what suggestions for improvement do you have, and so on.

Grading. Grading will be out of 100%. Roughly one third of the grade will be given for the database design. That is, if the database contains all the information required to implement the functionality described in this document, you will get 33% of the grade. We are not grading on the quality of the design yet. However, a simple design will make your life easier when you write the rest of the application. Roughly 55% of the grade will be given for the correct implementation of the queries, the processing of the results, and the responses to the user, as well as the quality of the HTML. Again, nothing fancy is required on the client side. However, the pages should look somewhat orderly, this is why using tables is a good idea. The final 10% of the grade will be given for clarity of your writeup.

Suggestions and Hints.

1. Start by creating the HTML pages and how they would look like. You can use HTML editors for this. Adobe Dreamweaver is a great one, but expensive. There are free ones that you can get online. Eclipse has a plugin for that too.

2. Identify inside each HTML page, what is the content that comes from the database and what is the static content.

3. Start by adding simple JSP code to your HTML file and by testing it.

4. Implement the different functionalities one at a time.

5. START EARLY!!!!

6. ASK QUESTIONS!!!