CIS 3308 Logon Lab

Lab Overview:

In this lab, your web application shall provide logon and logoff functionality and a "membersOnly" page that is only available to logged-on users.

Background Information:

The internet was created using the **REST design philosophy** which maintains that a server should only have its resources consumed ***while it is responding*** to a client request. Once the server responds to a client request, the server "forgets everything" about the client request. REST stands for Representation State Transfer and you can read a more about it here if you wish: https://en.wikipedia.org/wiki/Representational_state_transfer.

It might be great for the web server to continually forget what we have typed into a previous web page (like our user name and password), but that is not so great for the users – we don't want to have to re-enter their username and password for each page we visit. In the web, there are **several ways to persist data**:

1. **Page level persistence:** this is where a page posts data to itself. The page does this by using the **JSP implicit request object**. More specifically, we use the request.getParameter() method to extract user entered form data from the URL (request).

2. **Session level persistence:** this is where one page can store data (into the **JSP implicit session object** on the web server) and other pages can read this data. Each logged in user has their own session object. While this does not align with the REST philosophy, we do need to consume a little of the web server's resources so that users do not have to log in at every page. We use these two methods to get and put data (that persists until log out or time out): session.setAttribute() and session.getAttribute().

3. **Application level persistence:** like JSP Session level persistence, we can store data (into the **JSP implicit application object** on the web server) and other pages can read this data. With application level persistence there is one global application object that all pages and all users share. This might be a good place to store some global configuration settings for the web application, using methods: application.setAttribute() and applicaction.getAttribute().

4. **Cookies:** with cookies (similar to JSP session level persistence), one page can store data (on the client's PC/MAC) and other pages can read this data. This does not go against the REST philosophy, but it is very insecure and easily hacked. So, we may store some "user preference" type data in a cookie (so that the web server doesn't have to store it), but we would never store any sensitive information (like username/pw, log on status, authorization level) into a cookie. Also, cookies are read and written using client side programming, javaScript, which is not the focus of this course. Due to time limitations (and given it is not a secure way to store data), we will not cover cookies, except for maybe a demo in lecture.

JSP implicit objects are objects are available from any JSP page (pre-declared by tomcat/glassfish, the JSP application server software) but not accessible from java classes. In addition to the **request** and **session** JSP implicit objects, we have used:

- the **out** JSP implicit object - we used out.print to write HTML to a page.

And we need (in this lab) to use:

- the **response** JSP implicit object so we can response.sendRedirect() the user to an error page if they request a page for which they are not authorized and authenticated.

Functional Requirements:

1. For this lab, your web application shall have the following new pages:
   - **<u>logon.jsp</u>**: This page shall provide text boxes for user name and password.
     - If the user clicks submit with correct credentials, the page shall display a welcome message, otherwise it shall display an error message.
     - The input tag for password shall use type="password" (instead of type="text"), so that the letters of the password box show up like bullets (for security).
     - If there is a problem with the database, the page shall display a user friendly database message (like "database unavailable, please try later") followed by the real technical error message (so problem can be identified and resolved). To test this, run your logon page from home when you are not tunneled in.
     - Once your logon page is fully debugged, change the form's method tag (from "GET") to "POST". This will make is a little more secure (from spying eyes looking at the URL). Of course, we should also use https instead of http if we wanted to get really secure.
     - To facilitate our grading, please put a valid username and password into the textboxes intially (instead of "").

     OR -- or you can incorporate a log on area within **<u>index.jsp</u>**. If you choose to do this, you may have to work a bit harder to hide/show the logon area depending on user's logon status.
   - **<u>membersOnly.jsp</u>**: This page shall display some information, but only to logged-in users. If the user is not logged in, this page shall redirect them to an error page (**deny.jsp**, see below).
     - *Try to think up something to show your "members" when they are logged in, something that makes sense for YOUR particular web application.*
   - **<u>deny.jsp</u>:** This simple page shall just display an error message that explains why the user is not seeing the page they requested.
   - **<u>logoff.jsp</u>**: This simple page shall invalidate the session of the logged-in user.
     - *This page does not need to have a user interface. It can just invalidate the session then redirect to a more useful page (like logon or index page).*

2. Your headToContent.jsp (include file) shall provide a **<u>dynamic login/logoff link</u>** meaning that if the user is logged on, the page shall display a message that welcome's the user (showing some attribute that was pulled from the database during the logon process) and asks if they want to log off. Otherwise, they will be provided with a link to log on (or shown a log on area on the index page). However, the logon page (or index page if you implemented logon there) shall not show any message logon/logoff.
    - *By putting this logic in your include file, all pages can have the same behavior (just described) – UI code reuse.*
    - *Remember to put your "headToContent.jsp include directive" AFTER your JSP logic - on all pages. Otherwise, you'll experience behavior on some pages where the user has to refresh the page before the welcome message changes to reflect the users newly changed logged-on or logged-off status.*
    - *To avoid having a "logoff/logon" message on the logon page, pass a parameter to your top.jsp (as shown in the sample code).*

Design Requirements:

Your web application shall have a new package named "**modelX**" or "**model.X**", where X is similar to the name of your user table in your database. *The difference is that with "modelX", you just have one package/folder, whereas with "model.X" you actually have a package/folder (named X) inside of another package/folder (named "model").*

- In your modelX package, your web application shall have a class named "**StringData**". *Naming it this way will help you in later labs, so that your code will better align with sample code I will be providing to you.* This class shall have one public String variable for every field in your user table (even the non-string fields). It shall also have an extra public String variable that will represent a possible error message.
- Somewhere in your modelX package shall be a method that accepts a open DbConn object, an email address, and a password. This method shall return a populated StringData object if the credentials are good. It also needs to communicate a "not found" scenario as well as database exception message.
    - *Don't allow a possible database exception to "bubble up" to the JSP page – handle it with reusable class code. While debugging you could have any number of database exception messages (e.g., you may have a SQL syntax error), but once the code is debugged, you should just have the possibility of database unavailable (e.g., due to network or server outages).*

Other Requirements / Good Coding Style:

Follow all of the good coding style requirements as specified in the labs page (under "Requirements for All Labs and Project").

Labs Page/Blog:

For this lab and every lab, enter a blog into your labs page. Discuss things about your experience with this lab, like what you learned, what was hard/easy, etc. Link to the work for this lab (e.g., the logon, logoff, members only page).

Submission:

Publish your code, test your code, and submit a zip file of your WHOLE project into blackboard (includes all previous labs with everything still working).