CIS 3308 Logon Homework

Lab Overview

In this lab, you shall enhance your web application so that it provides logon and logoff functionality and a profile page that is only available to logged-on users.

When you click on “Account – Log on”, users shall see content like this:

![Logon Page]

After entering credentials and clicking the Log On button, the page shall either display a welcome message or an error message (DB error message or “user not found” message).

After clicking “Account – Log off”, users shall see content that indicates that the user is logged off (or an ajax error, for example, if they are no longer connected to the internet or if your code misspelled the name of the logoff API).

After clicking “Account – Profile” (if you are not logged in), the web app shall display a message like this:

![Profile Page]

But if the user WAS logged in, they should see all the fields of their Web User record that was stored into the JSP session when they successfully logged in (no database access).
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.

While it might be great for the web server to continually forget what users have typed into a previous web page, it is not so great for the users – they don’t want to have to re-enter their username and password for each page they visit. Adhering 100% to this REST philosophy, the web designers came up with the idea of COOKIES as a client side place to set and get data about each user. The only problem with cookies is that they are insecure – it is too easy for a hacker to steal, manipulate, and/or spoof cookie data. So, we will not focus on cookies in this class, other than perhaps having a lecture where I demo some cookie code (in JavaScript, all client side).

So, the designers of web server platforms (like JSP, PHP, .NET etc) added a “session” object (one object one per user) where you can set and get user attributes securely in the web server. Even though this goes against the REST philosophy, it enables a web application to securely “remember” a few things about each logged in user.

The JSP implicit session object is available only in JSP pages (like our Web APIs), not in java classes. It’s called implicit because it’s already pre-declared in any JSP page - we don’t have to explicitly declare it. We can use the following two session methods to store and retrieve data about a user. Each method is shown in a code example. For this sample code, imagine that you have declared a ShoppingCart java class and this is what you want to store into the session and pull it back out later.

```java
session.setAttribute(shoppingCarObj, "userCart"); // store the (populated) object into the session.
    // The object can be retrieved later (by any page) by asking for “userCart” (see next example).
    // You’ll need code like this in you Logon Web API

ShoppingCart theCartObj = (ShoppingCart) session.getAttribute("userCart");
    // Pull the object back out of the session and type cast it so you can access its properties.
    // You’ll need code like this in your Get Profile Web API
```

If you want to destroy the session object, you just use:

```java
session.invalidate();
    // You’ll need code like this in your Log Off Web API
```
Functional Requirements

- **Nav Bar:** To your web application, you shall add a drop down nav bar entry entitled “Account” with three submenu items below it: Log on, Log off, and Profile.

- **Log on:** If the user clicks “Log on” from the navigation bar,
  o two textboxes (a username /email address and password) and a “log on” button shall appear in the content area of the page.
  o After clicking the button (if the email address and password are found in the database), all of the attributes of the found user record shall be taken from the database and stored into the JSP session and a positive confirmation message shall be displayed on the page.
  o If the email address and password were not found in the database (or there was some database exception/error), an error message shall be displayed.

- **Log off:** If the user clicks “Log off” from the navigation bar, they shall see a message in the content area that tells them they are logged off (or an error message, for example, if they are no longer connected to the internet).

- **Profile:** If the user clicks “Profile” from the navigation bar, then if the user is logged on, the content area shall display their own user profile (all fields and values of their own web user record); otherwise (if they are not logged in), they shall see an error message indicating that they cannot see their profile because they are not logged in (or they might see an ajax error, for example if the Web API was misspelled or the internet is down).
Design Specifications

**Project Organization.** Your project shall be organized as shown below. New files for this week are indicated with red arrows. Please name the files exactly as we have them below – this will help us when we grade and enable us to help you if you run into any coding issues. You’ll have:

- a new html partial (contains Log On UI),
- a new javascript file (holds code for the Log On button),
- three new Web APIs (log on, log off, and get user profile) and
- a new java class Search (searches for user record based on user provided credentials).

![Projects Tree Diagram]

Design specifications for the Web APIs are described under the Suggested Approach section (see screen captures of sample outputs).
Suggested Approach

1. If you missed the lab activity for this week or you didn’t have time to finish it in lab, I suggest that you complete the tutorial that is linked from this week’s homework in the 3308 web page. This tutorial (about finding a webUser record based on a user entered id) demonstrates several important things that you need to know to complete this homework:

   - how to create user input areas and buttons in HTML (<input> tag, <button> tag)
   - how to access the values that users have entered into the input areas using JavaScript (get reference to the HTML element, then access its value attribute)
   - how to pass user input values to a server side Web API (you append them to the end of the URL)
   - how the Web API can extract the user input values from the URL (using request.getParameter("...")
   - How to inject user input (safely) into a SQL select statement (use the setString method of the java.sql.PreparedStatement object).

2. Start by working on your Web APIs. Using URL tampering, you should be able to test all three pieces of functionality log on, log off, show user profile (only if logged on). You test the Web APIs by running the JSP pages directly from NetBeans and the user interface is just the JSON responses provided by the Web APIs (install JSON Viewer plugin to Chrome if you have not done so already – makes it easier to read the JSON in the browser).

3. If you are working from a laptop or from a desktop at home, remember to tunnel in whenever you attempt to run a page that needs to access your Temple database. You’ll find a link to tunneling instructions listed under this week’s homework in the 3308 web page.
4. Start working on the **logonAPI.jsp** first. You learned from this week’s tutorial how to use request.getParameter to extract user input from a URL, but let me reiterate how that works. The JSP page needs to use code like this to extract values from the URL:

```java
String strEmail = request.getParameter("email");
String strPass = request.getParameter("password");
```

If you right click and Run your Logon API from NetBeans, you’ll see a URL like this (with no parameters tacked to the end):

```http
http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp
```

The two values shows above (strEmail, strPass) will both evaluate to null since they were not provided in the URL. Your Web API must test for null and provide an error message something like this – to let the client side programmer know that they needed to provide specific parameters in their URL:

If email and password are provided in the URL parameters expected by the Web API,

```http
http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=joe&password=joePass
```

then you might get other errors, like some database error (e.g., not tunnel in or bad DB logon credentials):

```
// 20180915091135
// http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=joe&password=joePass
public class logonAPI {

    "webuser Id": ",",
    "userEmail": ",",
    "userPassword": ",",
    "birthday": ",",
    "membershipFee": ",",
    "userRoleId": ",",
    "userRoleType": ",",
    "errorMsg": "Programmer error: must provide 'email' and 'password' as parameters in the ajax call."
}

http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=joe&password=joePass
```

```
// 20180915091135
// http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=joe&password=joePass
public class logonAPI {

    "webuser Id": ",",
    "userEmail": ",",
    "userPassword": ",",
    "birthday": ",",
    "membershipFee": ",",
    "userRoleId": ",",
    "userRoleType": ",",
    "errorMsg": "Problem getting connection:Communications link failure\n\nThe last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server."
}
```
Getting past a database error, it might be that the provided user email and password that do not exist in the database. If that is the case, the API shall show a message that indicates “not found”:

```
// 20180915092332
// http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=sally&password=p
{
    "webUserId": "130",
    "userEmail": "sally",
    "userPassword": "p",
    "birthday": "",
    "membershipFee": "",
    "userRoleId": "3",
    "userRoleType": "Member",
    "errorMsg": "" // user name and password not found.
}
```

BUT, if we provide the right URL parameters and provide values that ARE found in the database, then we should get a positive response like this that shows no error message and a fully populated WebUser record that matches the credentials provided in the URL parameters.

```
// 20180915092332
// http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=sally&password=p
{
    "webUserId": "130",
    "userEmail": "sally",
    "userPassword": "p",
    "birthday": "",
    "membershipFee": "",
    "userRoleId": "3",
    "userRoleType": "Member",
    "errorMsg": ""
}
```

Remember that when the correct parameters are supplied (with values found in the database), then the Log On Web API is (in addition to providing a positive message) also supposed to write the found Web User record into the JSP session object.

```java
session.setAttribute(webUserObj, "user"); // store the (populated) object into the session.
// The object can be retrieved later (by any page) by asking for “webUser” as input
// parameter to the session.getAttribute method.
```

Also, recall (from the Good Coding Requirements for all Homeworks from the 3308 web page) that JSP pages are supposed to be short and push as much code as possible down to reusable java classes. When I solved this problem, I put a method into model.webUser.Search that found and returned a WebUser object (passing in user email and password). If there was a database error, that message was placed in the errorMsg field of the WebUser object. If there was no db error and the record was not found, my method returned null. Decide how you want to write your find method.
5. Next, I suggest that you work on the **getProfileAPI.jsp**. This API does not need any parameters passed in the URL and it will not even access the database. It is simply supposed to check the session to see if anything has been stored in there.

   ```java
   WebUser userObj = (WebUser) session.getAttribute("user");
   // Pull the object back out of the session and type cast it so you can access its properties.
   ```

   The `userObj` in the code above will either have the value of null (if no one logged in or if a timeout occurred) or it shall hold the fully populated `WebUser` object that was found in the database at logon and placed into the JSP implicit session object. Here are two examples of output from the `getProfileAPI.jsp`. The first is when the `session.getAttribute` method returns null (because no one is logged in):

   ```json
   {
   "webUserId": ",
   "userName": ",
   "userPassword": ",
   "birthday": ",
   "membershipFee": ",
   "userRoleId": ",
   "userRoleType": ",
   "errorMsg": "Cannot provide profile - you are not logged on."
   }
   ```

   The second is when the `session.getAttribute` method returns a `WebUser` object (and your code type casts that and then converts it to JSON using the GSON jar code):

   ```json
   {
   "webUserId": "130",
   "userName": "sally",
   "userPassword": "p",
   "birthday": ",
   "membershipFee": ",
   "userRoleId": "3",
   "userRoleType": "Member",
   "errorMsg": ",
   ```

6. Next, you work on the **logoffAPI.jsp**. This API does not need any input parameters and it only needs to run one line of code:

   ```java
   session.invalidate();
   ```

7. Next, **test all three Web APIs** by logging on and getting the profile, then logging off and getting the profile. Verify that all three Web APIs function correctly BEFORE you start on any client side code.
8. Next, I suggest that you work on **client side Log On functionality**.

a. The Nav Bar “Log on” click shall invoke an ajax call that will simply place the **logon html partial** into the content area, similar to what you in your HomePage Homework – when you placed the homeContent.html and blogContent.html into the content area. This is an example of code you might put into your logonContent.html page (see below). Note that `<input type="password">` is just like `<input type="text">` except dots show in the box when the user types (instead of showing the actual characters typed).

```html
Email Address:  <input type="text" id="emailAddressId" />  
Password:  <input type="password" id="passwordId"/>  
<button onclick="logonFn('emailAddressId', 'passwordId', 'logonMsgId')">Log On</button>
<br/><br/>
```

Test that when you click on the Nav Bar “Log on” link, you do see the logon html partial in the content area.

b. Next, write the code for the **logon function** (that is invoked from the button shown just above). Define this function inside of logon.js. There are different ways to do this, but here is one:

```javascript
function logonFn (emailId, pwId, msgId) {
    var emailUserInput = escape(document.getElementById(emailId).value);
    var pwUserInput = escape(document.getElementById(pwId).value); // escape cleans user input
    var URL = "webAPIs/logonAPI.jsp?email=" + emailUserInput + "&password=" + pwUserInput;
    ajaxCall (URL, processLogon, processHttpError);
}
```

```javascript
function processLogon(httpRequest) { // this function defined inside of logonFn, local to logonFn
    var msg = "Successfully called the logon API (check console log for object contents). ";
    var obj = JSON.parse(httpRequest.responseText);
    console.log("Successfully called the logon API. Next line shows the returned object.");
    console.log(obj);
    if (obj.errorMsg.length > 0) {
        msg += "However, the Logon API supplied this error message: " + obj.errorMsg;
    } else {
        msg += "<br/>Welcome Web User number " + obj.webUserId;
        msg += "<br/> &nbsp; with Birthday: " + obj.birthday;
        msg += "<br/> &nbsp; and MembershipFee: " + obj.membershipFee;
        msg += "<br/> &nbsp; and User Role Id: " + obj.userRoleId;
        msg += "<br/> &nbsp; and User Role: " + obj.userRoleType;
    }
    /* I like to copy/paste the JSON code (below) so that I do not misspell any property names (above):
    "webUserId": "110", "userEmail": "bri", "userPassword": "no", "birthday": ",
    "membershipFee": ", "userRoleId": "1", "userRoleType": "Admin", "errorMsg": 
    */
    document.getElementById(msgId).innerHTML = msg;
}
```

```javascript
function processHttpError(httpRequest) { // this fn is also private/local to logonFn, good coding style
    document.getElementById(msgId).innerHTML = "Logon API call failed: " + httpRequest.errorMsg;
}
```
Now you should be able to test (all test cases) of your Logon API from your new user interface: not tunneled in (DB error), good credentials (found in database) and bad credentials (not found in database).

REMEMBER to place a script tag in index.html to reference logon.js or else none of your code in logon.js will be found...

c. Next, work on what happens when the user clicks on Get Profile from the nav bar. This should be similar to what you did for your DisplayData homework, the nav link click invokes an ajax call to the Get Profile API, providing two functions – one that writes to the content area whatever the Get Profile API responded, and an error function (that you probably already have) that would kick in, for example, if you lost your internet connection. If you need any JS code to support the Get Profile functionality, place it in logon.js

d. Next, work on what happens when the user clicks on Log Off from the nav bar. The nav link click invokes an ajax call to the Log Off API providing two functions – one for successful ajax call and one for failure. The only failure case should be unable to connect over internet to run the Web API. If you need any JS code to support the Log Off functionality, place it in logon.js

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 your experience with this lab - what did you find hard/easy, and what important concepts did you learn.

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).
Example Deductions

- **Submission:**
  - -9 if no **zip file** submitted into Canvas (by the due date).
  - -9 if no **pages published** by the due date.

- **Log On Functionality:**
  - -8: Log On does not work – if log on does not work, you cannot demonstrate get profile nor log off.
  - -3: get profile and/or log off does not work (only logged in users should be able to see their profile).

- **Blog:**
  - -1 if you don’t have a blog that links to a data model image.
  - Up to -1 for lack of **blog content** (where you tell what was hard/easy/important about this week’s HW).

- **Display Data:**
  - -1 if **web user** records not displayed.
    - -1/2: if there are no web user records that demonstrate that your code gracefully handles null non character fields (membership fee and birthday).
    - -1/2: if your web user records are not ordered by the first column shown in the HTML table.
  - -1 if **other** records not displayed.
    - -1/2: if your other data does not have a null date or numeric (integer or decimal).
    - -1/2: if your web user records are not ordered by the first column shown in the HTML table.
  - -2 if **assoc** records not displayed.
    - -1/2 if your associative data does not have a null date or numeric (integer or decimal).
    - -1 if your associative data does not include non-key fields from web_user and other.
    - -1/2: if your data is not ordered by the first column shown in the HTML table.
    - -1/2: if your data does not look professional. Use your judgment to combine fields (from database) into a single column of your HTML table to make your data appealing and easy to read.
  - Up to -2 if data in **HTML tables** are not formatted and/or not aligned properly (numeric should be right justified).

- **Layout:**
  - -0.25 if your name is not in the **footer**.
  - -1 if you have not used a **drop down menu** in your nav bar.
  - Up to -2 for lack of **professionalism/originality**. The page should look at least as nice as the sample provided for the Home Page HW (no text too close to visible edges, all text legible, use of subtle color scheme, fluid layout, no unwanted wrapping at any browser width, no large images slowing down page load). Also, your pages cannot overly similar to sample code nor classmate code.
  - Up to -3 if **JS/AJAX code not reused** (to invoke the home content, the blog content and the three new content areas: users, assoc, and other).

- **Coding Style / Design Specs:**
  - Up to -2 for what we notice when we “**View Source**”:
    - **HTML/CSS/JS syntax errors** which would show in red font from Firefox View Source.
    - **poor coding style**. Your code should be neat, properly indented, well named, and with no unnecessary code.
  - Up to -2 for **code organization** (not meeting design specs). Students show their project opened up in NetBeans.