CIS 3308 Logon Homework

Lab Overview

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

Client Side Functional Requirements

When users click on “Account” (or the Account icon), they shall see drop down options that include Log In, Profile, and Log Out. (Register will be implemented later.)

After clicking “Account – Log on”, users shall see a logon area with email address, password, and button.

After entering credentials and clicking the button, the page shall either display a welcome message (showing all the data about the logged on user as retrieved from the database) or an error message (ajax error message, DB error message, or “user not found” message). Below is an example of a successful logon message.
After clicking “Account – Profile”, the web app shall either show all the data about the logged in user or display an error message (ajax error message, DB error message or “no user logged in” message).

After clicking “Account – Log off”, users shall see a message that indicates that the user is logged off (or ajax error message – there is no chance for DB error message since the Logoff API does not need to access the database).

**Background Information – REST**

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 more about it here if you wish: [https://en.wikipedia.org/wiki/Representational_state_transfer](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.

**Background Information – JSP implicit `session` object**

JSP implicit objects are available only in JSP pages (like our Web APIs), not in java classes. They are called implicit because they are already pre-declared in any JSP page - we don’t have to explicitly declare them.

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 you want to store a ShoppingCart object 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.
// Whatever name you provided to method session.set is the name you must provide to session.get.
// You’ll need code like this in your Get Profile Web API
```

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

```java
session.invalidate();
// You’ll need code like this in your Log Off Web API
```
Background Information – JSP implicit request object

If you have a URL like this (with URL parameters email and password):

```
http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=sallyk@temple.edu&password=hi
```

Then you can extract the URL values (from a JSP page) using code like this:

```java
String strEmail = request.getParameter("email"); // would evaluate to “sallyk@temple.edu”
String strPass = request.getParameter("password"); // would evaluate to “hi”
```

You should have had some exposure to request.getParameter by doing the Lab Activity for this week.

Server Side Functional Requirements: Three Web APIs (server side, JSP pages)

logonAPI.jsp: takes two input parameters, e.g, email and password such as the URL shown above, then it
- Extracts the input parameters from a URL, a Web API (jsp page) using the getParameter method of the request JSP implicit object such as shown above.
- Invokes a method such as the one shown below. Store this method in a new class named model.webUser.DbMods (so, StringData refers to model.webUser.StringData). Your method does not have to be just like this, but the method needs to output 3 different things: found/webUser, not found, DbError.
- This Web API returns JSON that either indicates an error message (e.g., DB error or “user not found”) or it provides all the data of the web user record that has the given email address and password.

```java
package model.webUser;
import dUtils.DbConn;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class DbMods {
    public static StringData logonFind(String email, String pw, DbConn dbc) {
        StringData foundData = new StringData();
        if (email == null || pw == null) {
            foundData.errorMsg = "model.webUser.DbMods.logonFind Programmer error: email and pw must be both non-null."
            return foundData;
        }
        try { // prepare (compiles) the SQL statement
            String sql = "SELECT web_user_id, user_email, user_password, membership_fee, birthday, image, " + "web_user.user_role_id, user_role_type " + "FROM web_user, user_role " + "WHERE web_user.user_role_id = user_role.user_role_id " + "AND user email = ? and user password = ? ";
            PreparedStatement pStatement = dbc.getConnection().prepareStatement(sql); // compiles the SQL
            pStatement.setString(1, email); // Encode input param into the 1st ? of the sql preparedStatement
            pStatement.setString(2, pw); // Encode other param into the 2nd ? of the sql preparedStatement
            ResultSet results = pStatement.executeQuery(); // run the SELECT statement
            if (results.next()) { // Record found in database, credentials are good.
                return new StringData(results);
            } else { // Returning null means that the username & pw pair were not found in the database
                return null;
            }
        } catch (Exception e) {
            foundData.errorMsg = "Exception in model.webUser.DbMods.logonFind()" + e.getMessage();
            System.out.println("******* + foundData.errorMsg);
            return foundData;
        }
        // logonFind
    }
}
```
Note: starting next week, each of your model packages will have a class named DbMods. This class will eventually hold server side code related to inserting, updating, and deleting data for the given DB table.

- **logoffAPI.jsp**: takes no input parameters. Simply invalidates the JSP session object. See example code under the “Background Information” section of this document. Takes no input parameters and does not require a data connection object.

- **getProfileAPI.jsp**: takes no input parameters and does no DB access. Simply gets (and returns) the web user object that was stored in the JSP session object (if there is one there) or it returns an error message, e.g., “no user logged on”. See example code under the “Background Information” section of this document.

### Design Specifications

According to our JS naming convention, every JS file shall be named the same as the single function or object that is globally declared inside. So, if you have a nice reusable single function like ajax, it can be in its own JS file, but if you have several functions that are related and you want to keep them in the same JS file, you can create a JS object (named the same as the JS file), then you can add 2 or more public methods to that object. For example, you might have a JS file named logon.js that creates a single global object named logon, then defines two public methods (logon.UI and logon.findUser) – this is a good software design pattern that lets you bundle UI with associated functionality.

```javascript
var logon = {};

logon.UI = function (id) {
    var content = `    // back tick
    <div class='logon'>
      <br/>
      Email Address <input type="text" id="logonEmailAddress"/>
      &nbsp;
      Password <input type="password" id="logonPassword"/>
      &nbsp;
      <input type="button" value="Submit" onclick="logon.findUser('logonEmailAddress','logonPassword','msgArea')"/>
    <br/> <br/> <div id="msgArea"></div>
  </div>
  `;                        // closing back tick
  document.getElementById(id).innerHTML = content;
};

logon.findUser = function (emailId, pwId, msgId) {
    // ...
};
```

Note that there was no problem adding logon.UI as a function into the routing table from the index page...
**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 helps us when we grade and enables us to help you if you run into any coding issues. You’ll have:

- new JS components -- logon.js was provided as sample code and works to authenticate logon credentials. It calls WebAPIs/logonAPI.jsp.
- You’ll have to decide how to implement logoff and profile but model your code after the other components.
- I created ajax2.js which is more like the jQuery ajax call (you pass it a parameter object, more self documenting, it does the JSON.parse for you). But I left ajax.js in there in case you need it for compatibility. I recommend you migrate to ajax2.js and then you would not need ajax.js.
- three new Web APIs (**logonAPI.jsp**, **logoffSPI.jsp**, and **getProfileAPI.jsp**) and
- a new java class named **DbMods** in the model.webUser package where you will have a method that tries to find a web user based on user provided credentials.
Other Requirements / Good Coding Style:

Follow all requirements specified in the labs page (under "Requirements for All Labs and Project").

Blog Requirements:

As with all homeworks, you’ll create a blog this week in your blog content describing what was hard/confusing versus easy, what was important.

Also, provide direct working links to these web APIs:

- Logon – complete with valid URL parameters for a successful logon.
- Logoff
- Get Profile
- GetAll Users

Even if you were not able to get all of the User Interface pieces working for this week, you can get partial credit for creating the three new Web APIs.

Example Deductions

- Blog and Web APIs:
  - -3: we have no way to easily see what credentials are valid within your database (Display Users not working and/or no link to Get All Users Web API).
  - -3: no link to Logon Web API and/or this Web API not working
  - -2: no link to Get Profile Web API and/or this Web API not working
  - -2: no link to Logoff Web API and/or this Web API not working
  - -2: no blog discussion of homework (what was important, easy, or hard/confusing).

- User Interface and JS code
  - -3: missing and/or nonfunctional User Interface for Logging on.
  - -2: missing and/or nonfunctional User Interface for Getting Profile of Logged on User.
  - -1: missing and/or nonfunctional User Interface for Logging Off.

- Up to -3 for Violating Coding Standards specified in “Requirements for all HWs/Project”, such as
  - Not following JS file naming standards
  - Having unused/unnecessary code
  - Lack of indentation to indicate level of code nesting
  - Lack of comments
  - Poor variable naming

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).
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 creating a WebAPI that finds 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. Always write server code first. So that means 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) – without any client side code at all. You test the Web APIs by running the JSP pages directly from NetBeans and you’ll see what they output (the JSON) in the browswer. Make sure you have installed the Chrome JSON Viewer plugin 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 last week’s homework under the “Enhanced Instructions for DB Access Web Apps”.

4. The “Enhanced Instructions for DB Access Web Apps” instructions (under HW4 Display Data of the 3308 web page) also remind you to use System.out.println() statements to debug your server side (java/JSP) code and tells you where to find that output. in the 3308 web page.

5. Remember to use the “Enhanced Instructions for DB Access Web Apps” to install sample code (or to move a project from one development PC/MAC to another) and to publish your code. Now that you have java code under Source Packages, there is more to it.
6. 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. If the URL to invoke the logon API (passing email and password) would look like this:

http://localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=sallyk@temple.edu&password=hi

Then logonAPI.jsp would need code like this to extract the values from the URL:

```java
String strEmail = request.getParameter("email"); // would evaluate to “sallyk@temple.edu”
String strPass = request.getParameter("password"); // would evaluate to “hi”
```

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

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

Without any URL parameters, the strings (strEmail, strPass) will both evaluate to null. 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.

```java
1 // 2018915991279
2 // http://localhost:8080/3308_05_logon/webAPIs/logonAPI.jsp
3 {
4    "webUserId": ",",
5    "username": ",",
6    "userPassword": ",",
7    "birthday": ",",
8    "membershipFee": ",",
9    "userRole": ",",
10   "userRoleType": ",",
11   "errorMsg": "Programmer error: must provide 'email' and 'password' as parameters in the ajax call."
12 }
```

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

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 tunneled in or bad DB logon credentials):
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”:

```
localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=Joe&password=JoePass
```

BUT, if the URL provided the correct parameters with 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.

```
localhost:8080/3308_05_logOn/webAPIs/logonAPI.jsp?email=sally&password=p
```

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 obj webUser obj into the session under name "user".
// The object can be retrieved later (by any page) by asking for "user" 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 concise, pushing as much code as possible down to reusable java classes.
7. 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 is logged in or if a timeout occurred) or it shall hold the fully populated `WebUser` object that was found at logon time 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": "", "email": "", "password": "", "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": "138", "email": "sally", "password": "p", "birthday": "", "membershipFee": "", "userRoleId": "3", "userRoleType": "Member", "errorMsg": "" }
   ```

8. Next, 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();
   ``

9. Next, **test all three Web APIs** (URL tampering) 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.
10. Next, I suggest that you work on **client side Log On functionality**.

a. Remember that your HTML page needs to have a reference to each of your JavaScript files. The browser is unaware of JavaScript code that is in a file that is not referenced by a `<script>` tag – including a reference to your new account.js file.

b. The Nav Bar “Log on” click shall invoke a JS function that builds the Logon User Interface.

   In my solution, I created a JS file named *js/components/logon.js* which contained code for (1) creating the logon user interface and (2) code for handling the logon button click event. Generally speaking this code is structured as follows, some pseudo code / comments substituted for actual code.

   ```javascript
   var logon = { }; // JS naming convention, create single global object named the same as the file name
   logon.UI = function (id) { // create User Interface for Logon area.
     // input type="password" puts dots instead of letters in the textbox...
     var content = `<div class='logon'>
     <br/>
     Email Address <input type="text" id="...some ID"/>
     &nbsp;
     Password <input type="password" id="... some other ID"/>
     &nbsp;
     <input type="button" value="Submit" onclick="logon.findUser('Id of email address','Id of PW','Id of msg area')"/>
     <br/> <br/>
     <div id="... some other ID"></div>
     </div>``;
     document.getElementById(id).innerHTML = content;
   };

   logon.findUser = function (emailId, pwId, msgId) {
     // escape cleans user input and makes it safe to use in a URL (for ajax call).
     var emailUserInput = escape(document.getElementById(emailId).value);
     var pwUserInput = escape(document.getElementById(pwId).value);
     // build URL to make ajax call to logon Web API
     // success function for the AJAX call, displays logon message (error or success welcoming the user and
     // showing the user information about themselves).
   };

   c. Next, work on what happens when the user clicks on **Get Profile** from the nav bar. In my solution (which you do not have to copy), I created a file named *js/components/profile.js*. This code invoked the Get Profile Web API using an ajax call, then communicated the results to the user (showing all information about the logged on user or providing some kind of error message).

   d. Next, work on what happens when the user clicks on **Log Off** from the nav bar. The onclick even should invoke an ajax call to the Log Off API then (in the ajax success function), update the user interface to let the user know that the call was successful.