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. You do not have to use icons in your Nav Bar – you can use regular word links with Drop Down menus.

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 like this:

After entering credentials and clicking the Log On 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 – JSP implicit session object

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 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.

```
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:

```
session.invalidate();

// You’ll need code like this in your Log Off Web API
```

Background Information – JSP implicit request object

In the “Find by ID” tutorial (which is provided along with this homework on the 3308 web page), you learned how to extract data from a URL by using request.getParameter(). If you forget how to do this, there is sample code farther down in this document.
Server Side Functional Requirements

You shall add three Web APIs to your application this week, each with the functionality described below:

- **logonAPI.jsp**: takes two input parameters, e.g., email and password such as shown below:

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

  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.

  To extract input from a URL, a Web API (jsp page) uses the `getParameter` method of the `request` JSP implicit object such as what is shown below. The comments in the code below indicate the values as they would be evaluated given the above URL:

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

  The JSP page shall invoke a method named like “logonFind” that is to be stored in a new class named DbMods that is to be added to package model.webUser (so, StringData refers to model.webUser.StringData).

  ```java
  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: 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, "
                          + "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 = ? "
                          + "ORDER BY web_user_id ";
              PreparedStatement pStatement = dbc.getConn().prepareStatement(sql); // compiles the SQL
              pStatement.setString(1, email);
              pStatement.setString(2, pw);

              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 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;
          }
      }
  }
  ```
The example code above needs the following import statements (I like to import only what I need instead of doing something like import java.sql.*):

```java
import dbUtils.DbConn;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

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.

- **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.

### Client Side Functional Requirements

The client side functional requirements were already described in the overview (and with pictures 😊).

According to our naming convention, every JS file is named according to the single function or object that is globally declared inside. So, if you created a JS file named **account.js**, you’d define a single global object named “account” in there and then add whatever methods you need to this account object. In my solution, I had these methods:

```javascript
var account = {}; // define global empty object
account.logon = function (emailId, pwId, msgId) {
    // Must use “escape” to clean user input so security related errors don’t interfere with your code functioning
    var emailUserInput = escape(document.getElementById(emailId).value);
    var pwUserInput = escape(document.getElementById(pwId).value);

    // make ajax call, specifying callback function
    ajax("webAPIs/logonAPI.jsp?email=\" + emailUserInput + "\&password=\" + pwUserInput, processLogon, msgId);

    function processLogon(httpRequest) { // define callback function
        // ... you know what to do here: add info to the UI
    }
};

account.loggedOffMsg = function (httpRequest) {
    // ... you know what to do here: add info to the UI
};

account.profileMsg = function (httpRequest) {
    // ... you know what to do here: add info to the UI
};
```
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 helps us when we grade and enables us to help you if you run into any coding issues. You’ll have:

- a new html partial that contains the Log On UI named `logonContent.html`
- a new javascript file named mine `account.js` that holds all the client side code related to logging in, logging off, and getting the user’s profile
- 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.
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 browser. 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:

   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:

   Programmer error: must provide 'email' and 'password' as parameters in the ajax call.

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):

   Problem getting connection:Communication: limit failure
   The last packet sent successfully to the server was 8 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”:

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.

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.

```
session.setAttribute(webUserObj, "user");  // store webUserObj 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. 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 login find method.
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.

   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):

   ```
   // 20180915105546  
   // http://localhost:8080/3308_05_logOn/webApps/getProfileAPI.jsp
   
   {  
   "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.

   ```
   // 20180915092232  
   
   {  
   "webUserId": "130",  
   "userEmail": "sally",  
   "userPassword": "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:

   ```
   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 an ajax call that will simply place the **logon html partial** into the content area, similar to what you did 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).

   Email Address:  
   <input type="text" id="emailAddressId" />  
   Password:  
   <input type="password" id="passwordId"/>  
   <button onclick="account.logon('emailAddressId', 'passwordId', 'logonMsgId')">Log On</button>

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

   Note: students with server side web development experience might be tempted to place the above code inside of an HTML `<form>` tag, but this would cause problems. We do not want the button to submit to a server side page so **do not use any form tags this semester** – or if you do (because maybe you wanted to take advantage of HTML5 validation), change the button tag to be

   `<input type="button" onclick="account.logon('emailAddressId', 'passwordId', 'logonMsgId')" value="Log On"/>

   c. Next, write the code for the client side **logon function** (that is invoked from the button shown just above). There was some started code under “Client Side Functional Requirements”

   d. Next, work on what happens when the user clicks on **Get Profile** from the nav bar. The onclick event should invoke the get profile Web API using an ajax call. The callback function should update the user interface to let the user know the data that was returned about the logged in user (or an error message).

   e. Next, work on what happens when the user clicks on **Log Off** from the nav bar. The onclick event should invoke an ajax call to the Log Off API. The callback function should update the user interface to let the user know that the call was successful (or possible ajax error message, e.g. if internet went down).

**Other Requirements / Good Coding Style:**

Follow all requirements specified in the labs page (under "Requirements for All Labs and Project") – including creating a blog about what you learned and what you found easy/hard.

**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).