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

REST is a design philosophy with this goal: 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. The internet was created using REST. REST stands for Representation State Transfer and you can read more about it here if you wish: https://en.wikipedia.org/wiki/Representation_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. To adhere to this REST philosophy, the internet provided 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. We should try not to put too much data into server side session objects, so that our web application can scale without too much trouble.

Background Information – JSP implicit objects

Every JSP page has “JSP implicit objects” which simply means that these objects are pre-declared (implicitly) and available to the page. Here are some main JSP implicit objects:

- **out**: we have been using this JSP implicit object to output to the JSP page, e.g., out.print().
- **request**: we use this object to extract input parameters from the URL that invoked a Web API, e.g., request.getParameter("paramName").
- **session**: we use this object to securely store (server side) information about each user, e.g., session.setAttribute() and session.getAttribute(), and session.invalidate()
- **application**: if we have any web application global information we want to securely store (server side), we could use application.setAttribute(), application.getAttribute().
**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.

**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( shoppingCartObj, "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
```
Server Side Requirements: Three Web APIs (server side, JSP pages)

**logonAPI.jsp** shall take two input parameters, e.g., email and password such as in the URL shown below. It shall

- Extract 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.

If you have a URL like this:

```
http://localhost:8080/projName/webAPIs/logonAPI.jsp?email=sallyk@gmail.com&password=hi
```

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

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

If either of the URL parameters evaluate to null, provide a user friendly message to the programmer telling them the URL parameter names expected by the Logon Web API.

- Invoke a Java method (in model.webUser.DbMods) that searches for a web user record that has the given email address and password. This method shall return a StringData object representing three use cases: web user found, web user not found, and DbError.

- If logon was successful, the Web API shall write the StringData object (of the logged on web user) to the JSP implicit session object. This is so that any subsequent server side code can check to see if a user is logged on (and even know what is the user’s role, although our application is simple and does not differentiate functionality based on the role of the logged on user).

    ```java
    // put object into the session, giving it a name for later reference.
    session.setAttribute(webUserObj, "loggedOnUser"); // webUserObj is a populated StringData object
    ```

- This Web API shall output JSON code of the web user object just mentioned (same three cases: found, not found, db error).

**getProfileAPI.jsp**: *takes no input parameters* and does no DB access. Simply gets (and outputs JSON of) 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”.

```java
if ( session.getAttribute("loggedOnUser") != null ) {
    StringData loggedOnWebUser = (StringData) session.getAttribute("loggedOnUser");
    // Get the object back out of the session by supplying the name you provided when you put it in.
    // Returns null if there’s no object by that name in the session.
    // Must type cast the "plain object" that’s extracted from the session.

    out.print(loggedOnWebUser);  // this is an example of outputting a JSON string.
}
```

**logoffAPI.jsp**: takes no input parameters. Simply invalidates the JSP session object. Takes no input parameters and does not require a data connection object. So, that an AJAX call will not fail, the logoff API must out.print the JSON of some object (perhaps a newly instantiated and “empty” StringData object would be easiest to out.print). But the client side really does not need any information, other than the fact that the Web API did complete and did provide some output.

```java
session.invalidate( );
```
Client Side 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 the file. So, a nice reusable single function like `ajax` can be in its own JS file, but if you want to keep several related functions together, you could create a global object named the same as the file, then add the functions as public methods.

Your web application shall contain a JS file named “**account.js**” that creates a global object named “account”, then adds three public methods (functions) to that global object:

```javascript
var account = {};  
account.logon = function () {
    // ... build the user interface including the logon button with onclick handling to invoke AJAX call

    // here's how you create the password textbox that has dots instead of showing the input.
    var passwordInput = document.createElement("input");
    passwordInput.setAttribute("type", "password"); // so it shows dots not characters

    account.getProfile = function () {
        // ... build a message area, make an AJAX call to Get Profile API, display results
    }

    account.logoff = function () {
        // ... build a message area, make AJAX call to logoff API, display results
    }

    Here's how you would reference these functions in your routing table:

    myRoutes['#/logOn'] = account.logon;
    myRoutes['#/logOff'] = account.logoff;

When writing my solution code, my logon function had code that built the user information. When writing the getProfile code, I copy/pasted that code to getProfile. But since I’m not a big fan of the “copy/paste” software design philosophy, I created a function that could be called twice.

    account.buildProfile = function (userObj) {
        var msg = "";
        if (userObj.errorMsg.length > 0) {
            msg += "<strong>Error: " + userObj.errorMsg + "</strong>";
        } else {
            msg += "<strong>Welcome Web User " + userObj.webUserId + "</strong>";
            msg += "<br/> Birthday: " + userObj.birthday;
            msg += "<br/> MembershipFee: " + userObj.membershipFee;
            msg += "<br/> User Role: " + userObj.userRoleId + " " + userObj.userRoleType;
            msg += "<p><img src='" + userObj.image + "> "</p>";
        }
        return msg;
    }

    BUT... function buildProfile really should be private because it does not need to be called from any other context.
If you want buildProfile to be a private function that can be shared by the account functions, you can use a JavaScript IIFE (stands for Immediately Invoking Function Execution, pronounced IF-ee). Like its name suggests, an IIFE runs once and only once, when the page is first rendered. To create an IIFE, you define an anonymous function (no function name) the normal way:

    function () {...}

then call that function right away (adding left/right parentheses after the definition):

    function () {...} ();

but then wrap the whole thing in parentheses (to prevent ambiguity):

    ( function () {...} () );

Applying an IIFE in our context would look like this:

    var account = {}; // globally available object

    (function () { // This is an IIFE immediately invoking function execution

      account.logon = function () { // public method of global object
        // ... build the user interface including the logon button with onclick handling to invoke AJAX call
        
        // Here's how to build a password style text box showing dots instead of characters...
        var passwordInput = document.createElement("input");
        passwordInput.setAttribute("type", "password"); // so it shows dots not characters
        
        // ...
      }

      function buildProfile (userObj) { // NOW PRIVATE, can be called by any of the account functions...
        var msg = "";
        // ...
        return msg;
      }

      account.getProfile = function (){
        // ... build a message area, make an AJAX call to Get Profile API, display results
      }

      account.logoff = function (){
        // ... build a message area, make AJAX call to logoff API, display results
      }

    }) ();

}
Project Organization. Your project shall be organized as shown below.

New files for this week are indicated with red arrows. Please name the files 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 JS file – **account.js** which shall contain client side code for logon, logoff, and getProfile. Each of these three globally available functions shall be “Single Use Components” that take no input parameters and return a div that can be injected into the content area by routing. If you want to create an IIFE, that is fine. Otherwise, you can copy/paste the build profile code or have it be a public function. It’s more important that you understand your own code than it is for the code to be perfect at this point. You can incorporate IIFEs in a later assignment and maybe circle back to improve this later.
- 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. Going forward, DbMods (in model.WebUser) will contain any insert/update/delete code related to the web_user database table.
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 (needs no URL parameters)
- Get Profile (needs no URL parameters)
- List All Web Users

If you could not 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) along with your self assessment (based on the above deductions).
Suggested Approach

1. If you missed the **Lab Activity** for this week, I suggest that **you do that first**. (Remember to follow Enhanced Instructions to Install Sample code.) This Lab Activity demonstrates several important concepts that you need to complete this homework:
   - How to inject user input (safely) into a SQL select statement (use the setString method of the java.sql.PreparedStatement object).
   - How the Web API extracts parameter values from the URL (request.getParameter("paramName"))
   - Review of how to create textboxes and buttons in HTML: <input> tag, <button> tag.
   - Review of how to access the values entered into the textboxes: using javascript, get reference to the HTML input element, then access its value attribute of the input element.
   - How to pass user input parameters to a server side Web API (you append the parameters to the end of the URL that you use in your AJAX call and you should “escape” the input so that the server does not refuse to respond to the HTTP request of the AJAX call, for security reasons)

2. **Always write server code first.** There’s no way to make the client side work if the server side does not...
   a. After the Lab Activity, start by writing the **Logon API**. As you did in the Lab Activity, start from the “bottom up”, making the find java method, then write the Web API, then test that with URL tampering. **Remember that you need to be tunneled in when developing from home.** Once you are getting the right output from testing (web user found and not found, maybe even database error when not tunneled in), add logic to your JSP page, to store the web user object into the JSP session object. This logic shall run only when the logon is successful.
   b. I suggest you next write the **Get Profile API** which does not take any input parameters. It simply checks the JSP implicit session object and returns what is there. If there is nothing in the session, however, the object extracted (by getAttribute) will be null, so the JSP page needs to check for that. If the object is null, you probably should return a StringData object with all values “” (empty string) but with an error message saying no one is logged in.
   c. Write the **Logoff API** which should be very simple. It just needs to invalidate the session object (as shown below), but it should out.print the JSON of a valid object (maybe an empty StringData object), otherwise your AJAX call (client side) will fail trying to turn the page output into a JavaScript object.
   d. **URL Tamper** to make sure that your three web APIs work. Log on, get profile (should show you the web user that’s logged on), log off, get profile (should give you a message that no one is logged on).
   e. If you are having any problems with server side code (Java/JSP), **debug your code** with System.out.println() statements. You find this output in the Glassfish Server Log (in NetBeans, click on Window – Output and find the Glassfish Server tab in there).

3. Then start writing **Client side code: Logon, Logoff, Get Profile.** You should have enough sample code to get you started from the lab activity.

4. Even when testing client side code, you need to be **tunneled in** (since client side code calls your server side code using local/relative links). You’ll find a link to tunneling instructions listed on our class web site, under the section entitled “Enhanced Instructions”.

5. Remember to use the “Enhanced Instructions for DB Access Web Apps” to **publish** your code. Now that your web application has server side Java/JSP code, there is more to publishing.