CIS 3308 Ajax Search Lab

In this lab, you will create three Web APIs (JSP pages) that will "out.print" data in JSON format (JavaScript Object Notation). You will then create "search.html", a page that looks and functions like your search.jsp, except that it will only access data by making AJAX (Asynchronous Javascript And Xml) to Web APIs (that you create). At page load time, your HTML page will call the two select tag APIs to generate the select tags for your user and for your other table. When the user clicks the search button (button click event, not page submit), your HTML page will call an API that returns associative data (joined with user and other), based on the users search criteria.

Functional Requirements

1. **userSelectTag_API.jsp**: shall output JSON data that contains all the names and ids from your user table (plus a possible database error message).

   Hints:

   - When a JSP page out prints JSON (instead of an HTML user interface), it needs a different "page content type" - first line of code.
   - Import from the GSON jar file (code that can convert to/frome JSON) - second line of code.
   - Example of converting a java object (personList) to JSON format - last two lines of code.

   ```
   <%@page contentType="application/json; charset=UTF-8" pageEncoding="UTF-8"%>
   <%@page language="java" import="com.google.gson.*" %>
     . . .
     Gson gson = new Gson( );
     out.print(gson.toJson(personList).trim( ));
   ```

2. **otherSelectTag_API.jsp**: shall output JSON data that contains all the names and ids from your "other" table (plus a possible database error message).

3. **dataSearch_API.jsp:** shall accept criteria as input (from the URL) and then output JSON data that represents rows from your associative table (joined with user and "other) that match the criteria.

4. **search.html**: shall meet all the functional requirements of the earlier search lab. It shall have the same look and feel as search.jsp.

   Hints:

   - To easily create search.html to have the same look and feel as search.jsp, run search.jsp, then copy all the code from View Source, then paste into a new html page.

Design Specifications

1. **search.html**

   a. All of the javaScript code shall be placed inside of a <script> tag just before the closing body tag </body>.

   b. At page load time, search.html shall make two asynchronous AJAX calls to the SelectTag APIs. In the callback functions, javaScript code shall build two select tags (user and other) from the returned JSON data.

   c. The onclick event of the search button shall make an asynchronous AJAX call to the DataSearch API, passing in all of the user's search criteria (in the URL). In the callback function, javaScript code shall build a HTML table from the returned JSON data.

   Hints:

   - You can build a URL (using javaScript in the HTML file) that might look like this:

     ```
     var url = "dataSearch_API.jsp?customer=3&product=12&highPrice=20";
     ```

     and the JSP code can get those values by calling request.getParameter( ) three times.

   - Or, you might consider "bundling up" (using javaScript in the HTML file) these attributes like this:

     ```
     var criteria = { }; // declare and initialize empty object, in your HTML file (javaScript code)
     criteria.customer = 3;  // get this value from one of the select tags
     criteria.product = 12;  // get this value from the other select tag
     criteria.highPrice = 20; // get this value from a textbox
     var jsonCriteria = JSON.stringify (criteria); // a function that's built into the browser
     //  jsonCriteria would contain:     { customer:3, product:12, highPrice=20 }

     var url = "dataSearch_API.jsp?json="+jsonCriteria;
     // url would contain:   dataSearch_API.jsp?json={customer:3, product:12, highPrice=20}
     ```

     and then, your jsp page (dataSearch_API.jsp) can convert from JSON to POJO (Plain Old Java Object) like this:

     ```
     <%@page contentType="application/json; charset=UTF-8" pageEncoding="UTF-8"%>
     <%@page language="java" import="com.google.gson.*" %>
        . . .
        String criteriaStr = request.getParameter("json");
        Gson gson = new Gson( );
        Criteria criteriaObj = gson.fromJson(criteriaStr, Criteria.class);
        // you would want the properties of java class "Criteria" to be named the same as the
        //  properties in your javaScript code (e.g., customer, product, and highPrice).
     ```

2. Your web application's java/JSP code shall reference the GSON jar file to convert any POJOs (Plain Old Java Objects) to and from JSON.

Programming Style

- Follow all the programming style requirements listed in the 3344 web page (good naming, good code organization - MVC, minimal JSP code, java code reuse, proper usage of database connection objects, no unused code, etc).

Labs Page (Blog)

- Your labs page (as always) shall include a blog for each lab, explaining what you learned and linking to your work.

Homework Submission

- After getting your code to work locally, publish it to cis-linux2 and test it.
- Submit a zip file of your whole project to blackboard.
- Make sure that you have a link from your labs page to a screen capture of your data model (that you created using mySqlWorkbench).

Suggested Approach

1. If you did the lab activity, you should have learned how to:
    a. write and debug simple javaScript program using the Chrome Debugger (press F12 from Chrome):
        i. the Chrome Console for javaScript error messages and display of elements written using console.log to print debug messages to the Chrome console, and
        ii. the Chrome Element tab which can help you learn about the current properties of DOM elements.
    b. make ajax calls (o a third party Web API provider), and
    c. build Document Object Model elements (like <p>, <img>) from the JSON data (JavaScript Object Notation) that was provided by the Web API call.

    If you did not do the lab activity, visit my tutorial page for javaScript (which links to selected concepts from w3Schools). Google to learn about JSON and the use of the Chrome debugger.
2. Study the explanation of the JSON data format (you can go directly to W3Schools, or follow the link from this lab within the 3308 labs page).
3. Extract and study the sample code for this lab and install it into a new NetBeans project.
4. Install the JSON View plug-in for chrome (just google it). Tunnel in. Run one of the Web APIs from the sample code (just like any other JSP page, just right click and run). You should see well-formed JSON in your browser.

5. Study the PersonSelectTag_API.jsp page from the sample code and note that the first line of any Web API JSP needs a different contentType (different from the other, HTML user interface JSP pages we have been writing). It needs to be this:

```
<%@page contentType="application/json; charset=UTF-8" pageEncoding="UTF-8"%>
```

   Then begin working on **userSelectTag_API.jsp**.

   - You need to be tunneled in for this (because you are accessing a database).
   - You can copy in the SelectOption and SelectOptionList classes from the dbUtils package of the sample code.
   - You'll have to write a method that is similar to the code you wrote in the display data lab, to output a string containing the HTML code for a table full of user data. In the new method, in the "while results.next()" loop, you'll add a SelectOption to a SelectOptionList, instead of adding a <tr> tag to a <table> tag. Do not write this code right in the JSP page, write a java method that returns a SelectOptionList to the JSP page.
   - Carefully compare the output of your userSelectTag_API.jsp with the PersonSelectTag_API.jsp – just to be sure that you have a well formed JSON string (JSON View should help you with that, by identifying any syntax errors).

6. Run your old search.jsp page, then copy/paste its "View Source" into a new HTML page named search.html. We are mostly looking for the overall look and feel (and we cannot use JSP include statements in an HTML file).

   - You can delete the HTML table that shows associative data.
   - You can delete the user select tag since you will be building a user select tag using client side code in this lab. I would leave the other select tag there temporarily.
   - You can delete the form tag (<form> and </form>, not its contents) because you will not be posting form data to any server side JSP page.
   - You can delete the submit button (input type="submit", because we will not be posting form data anywhere) but you do need a search button (input type="button").

7. From the AJAX_Search.html (sample code) try to copy (into your new search.html) ONLY what is needed to

   - make the AJAX call to userSelectTag_API.jsp and
   - create a well formed select tag (dynamically, in the browser, using javaScript) that you can see on the page at page load time.

8. Repeat what you did for userSelectTag_API.jsp to create otherSelectTag_API.jsp. Delete the "other" select tag and proceed to have javaScript dynamically create this select tag after calling the API. At page load time, you should see two well formed select tags (user and other).

9. Create an enhanced StringData class that represents an associative record joined with user and "other" – you only need to create properties for the fields you plan to show in search.html (plus the ever useful "record level error message").

10. Create a StringDataList class that contains an array of these (enhanced Associative) StringData objects. The StringDataList object also needs a (list level) database error field so that it can communicate issues to the HTML page.

11. You should already have a search method that you used for search.jsp. This search method ran a SQL SELECT statement on your associative table joined with user and "other". This method accepted search criteria as input and included the search criteria in the WHERE clause of the SELECT statement. You'll need a searchJSON method that is very similar to the previous lab's search method except that the new method should return an enhanced StringDataList object (built by adding StringData objects from each row of the result set of the Select statement).

12. Create the DataSearch Web API JSP page. Like the sample code, it will need to declare a database connection object and call a searchJSON method. At first, you can pass all "" empty strings (to represent no filtering) to the searchJSON method. Run the code directly to the browser and make sure that the JSON is well formed. You should see all the rows from the result set, converted to a JSON string, well formed in your browser.

13. You can add an onclick event to your button in search.html:

    ```
    <input type="button" onclick="jsFunctionName()"/>
    ```

    the javaScript function would need to call your DataSearch API and the callback function should create a HTML table inside the content area of search.html. There is pretty good sample code for this (and if you did the lab activity, you should be pretty comfortable with creating and appending DOM elements). At this point, we are not trying to pay attention to any user entered filter values.

14. Next, add filtering. When your search.html page makes the AJAX call to DataSearch API, it needs to append the user entered values. If you have a textbox with id "highPrice", then you will find it's value (in javaScript) like this:

    ```
    document.getElementById("highPrice").value

        // or, if you have defined a function like $...

    $("highPrice").value
    ```

    The Data Search API (JSP page) will have to do a request.getParameter() for every URL parameter. For example, if the search page sends this URL:

    ```
    dataSearch_API.jsp?customerId=3&productId=12&highPrice=20
    ```

    your JSP page needs three statements like this:

    ```
    String custId = request.getParameter("customerId");
    ```

    Although we would expect search.html to send all three parameters (with "" as value if not selected), it would be safest (while debugging) to test each of the request.getParameter() values and replace null with "" emptry string – to avoid the dreaded java null pointer exceptions. Then you can start passing the actual user filter values to the searchJSON method and see if you are getting the desired filtering when you view the JSP page in the browser through JSON View. If so, your search.html page should work just fine !!!