Web API Lab

In this lab, you shall produce four deliverables in folder “07_webAPIs”. The first two deliverables should be pretty much done for you in the sample code.

1. A server side Web API (named listUsersAPI.jsp) that selects all the columns of all the rows from your web_user database table (ordered by id) and prints this in JSON format.

```java
// http://localhost:8080/3344_07_webAPI/07_webAPIs/listUsersAPI.jsp

{ 
  "dbError": "",
  "webUserList": [
    { 
      "webUserId": "1",
      "userEmail": "donald.j.otto@temple.edu",
      "userPassword": "p@ssw0rd",
      "birthday": "03/24/1994",
      "membershipFee": "$2,500.00",
      "errorMsg": ""
    },
  ]
}
```

2. A (client side) html page (named listUsers.html) that invokes your listUsersAPI using the AJAX technique and javascript (no jQuery) and modifies the DOM, creating an HTML table (adding rows and columns of data, from the JSON string).

![Table](image-url)

The next two deliverables you shall write yourself.

3. A server side Web API (named listOtherAPI.jsp) that selects all columns of all the rows from your “other” database table (ordered by your descriptive field) and prints this in JSON format.

4. A (client side) html page (named listOther.html) that invokes your listOtherAPI using the AJAX technique and javascript (no jQuery) and modifies the DOM, creating an HTML table (adding rows and columns of data, from the JSON string). For this page, you’ll have a column of image URLs. Create an `<img>` tag inside of the table data element `<td>` and set the `src` attribute to be the fully qualified URL that you read from the database. Make sure that you style all the images so that they have the same (reasonable) width). In the database, try not to reference images that are huge (will result in slow loading of page). It is OK to have some repetition in images, but you should have at least 5 or so unique images in your other database table.
You shall then add a "Web API" blog to your labs page and link to all four of the above files - this will let us see the data from your two database tables (1) nicely in a HTML table and (2) raw JSON formatted data.

**Web API**: API stands for Application Programming Interface (essentially a method call that has no user interface, a method that is exposed so that it can be called by external code). A Web API then is a server generated page (e.g., JSP page or PHP page or aspx page) that is only data (e.g., JSON format), has no HTML user interface.

**Project Organization**

For this lab, I recommend that you work in a new web application project until things are working, then copy in all your previous labs. Test everything test locally before you publish. Whenever you get to some milestone (of something working), make a copy of the whole NetBeans project folder (e.g., from file manager - name it so you know what’s working) and store this outside of NetBeans in some backup folder.

The diagram below shows you how to organize your Java Code. If you create a new project from the sample code for this week, your code should be organized correctly. In the red boxes below, I have tried to outline where you will need to make your changes to code for this week’s homework.

Folder for this week’s HTML and JSP files, including a subfolder with JS code. You can use this JS sample code (but study it !!!)

You may want to create a new style sheet for your “tabular data”.

You should not have to modify the classes in this package except for changing the database connection string credentials to be your own.

You should not have to modify the classes in this package since you created a web_user table exactly according to design specs.

Name this package substituting the name of your “other” database table for “xxx” (do not name this package “model.xxx” or “model.other”). Modify the StringData class to have property names that match the names of the columns in your “other” database table. StringDataList should just need a couple of changes (for example, “webUserList” would not be a well named identifier).

Need the GSON jar file (converts POJOs <-> JSON) and the MySql database drivers. “Enhanced installation instructions” explain how to add these in.
**Note:** (Almost) fully implemented sample code was provided this week, simply because there are so many new concepts (including server side programming which is not the focus of this course). Please DO NOT blindly use the sample code, modify it, and submit it as your work. If you do, chances are that your quiz and exam grades will reflect your lack of understanding. Please dig in deep and learn as much as you can from this week's lab.

**Functional Requirements**

1. When you did your database homework, you were required to have certain nullable non-character fields for your web_user table and for your “other” table. You were required to have at least one record that has null in all nullable non-character fields and at least one record that had all fields populated (none null). Your Web APIs and your HTML pages need to display all records with null values being displayed appropriately (empty string) - without any exceptions being thrown.
2. Extra credit (10%): if you can incorporate “clickSortable” data tables in the two HTML page like you did in your advanced JS homework.

**Design Specs**

Good programming style dictates that your listOtherAPI.jsp page should not have a lot of code in it. It should call a method in a class in the View package. This method shall specify the SQL select statement and pass back a nice StringDataList to the JSP page.

**Program Style Requirements**

Check the "All Labs and Project Requirements" section of the 3344 labs page.

**Layout Requirements**

For the two HTML pages, create a layout that is the same as either your home page or your blog page (they both should have a similar look and feel to each other anyway).

**Submission**

- Test locally (making sure everything still works from previous labs). This means all of your previous labs copied into a single NetBeans project. Test all links and all pages.
- Then publish and test what you published.
- Then submit a zip file into blackboard by the due date/time.
- When you demo your code to the TA, be prepared having the local project open so that he can check the naming and code style in the java classes.
How to Debug Java/JSP Code

In your other java classes, you have used System.out.println() to print debugging messages to the Console. This is the technique that you should use in this lab as you work on getting your JSP page to print valid JSON to the browser (with help from your java classes). System.out.println() statements are not printed into the web page, but they are viewable in the "Glassfish Server" log tab within the Output Window (lower right of NetBeans UI). If you work on a PC, you probably see this server log already (or can easily get to it from the NetBeans menu by selecting "Window – Output"). On the MAC, you can still get to the Glassfish Server log, but you need to

- click on the Services tab (upper right, where you are used to seeing your project navigator),
- open up the "Servers" tree element, then
- right click on "Glassfish Server" and select "View Domain Server Log".

Notice that in the Glassfish Server log of the following screen capture, you are seeing messages like "JSP Page ready to search for country with A" – this was generated by System.out.print() statement in the JSP page. This is followed by "Searching, for countries that start with A" which was generated from one of my java classes.
Suggested Approach

1. **Set Up Sample Project.**
   a) Begin by creating a new project that has just the sample code (from the zip file). Remember that you must use the “enhanced installation instructions” since our projects now have server side code (Java/JSP).
   b) After tunneling in, test that this project works. Look in the HW assignment for tunnel instructions.
   c) Make sure you have installed JSON View for chrome so that the JSON data is easier to read and understand in the browser.
   d) Once you are sure that the sample project works, leave it unmodified so that you will always have working code as an example. Remember that anytime you are running any database access code, you always must be tunnelled in if you are developing from home or from your laptop. (You do not have to be tunnelled in, of course, to run your published pages.)

2. **Set up HW Project.**
   a) Create a second new project from the sample code and use this as your working project for your homework for this week. Test that this project also works. Then change the database credentials (in dbUtils/DbConn) to be your credentials and then test/debug first the listUsersAPI.jsp then the listUsers.html.
   b) Rename model.xxx package (to an appropriate name for your “other” database table - but don’t name it “other”). Modify its StringData class so that its properties are appropriate to hold the values from the columns of data in your “other” table. Modify its StringDataList class renaming “webUserList” (which is no longer appropriate).
   c) Create a method in a class in the view package which will be similar to view.WebUserView.getAllUsers().
   d) Start working on listOtherAPI.jsp. It will be similar to listUsersAPI.jsp.
   e) If you are having trouble with compiler errors, you can try initially to just extract id and descriptor field, then add in the other fields.
   f) Once you have fixed all the compiler error messages, you can then run listOtherAPI.jsp directly. When you do this, you’ll likely see errors printed in the page. The first type of error you will likely run into is a database error such as syntax error in SQL select statement or incorrect DB credentials or you forgot to tunnel in. After resolving any database error, check carefully for record level errors such as trying to format a date as an integer, or a null value that throws an exception. Use System.out.println statements described in the previous section of this document to debug any java/JSP code.
   g) It is OK to use sample code as a starting point, but DON’T BE SLOPPY. Rename anything that needs to be renamed and if this causes an error, then that is a good opportunity to learn.
   h) Once you get listOtherAPI.jsp, begin working on listOther.html. It should be similar to listUsers.html, but you must modify field names to be appropriate for the new use. Remember to use console.log to debug your javaScript code. If you console.log an object (by itself), you can open up the object in the chrome console to see all of its properties.

3. **Back Up Often.**
   Whenever you get something to work, make a copy of the project folder (name it so you know what works in it) and store it somewhere).
4. **Merge in Code from Previous Homeworxs.**
   Once you are satisfied that you have completed this week’s homework requirements, back up your working project then merge in the code from your previous homeworks. As I have been stating all along, you are to have a single Web Application that has all of your homeworks working inside. It is pretty easy to merge client side code (what’s in your “Web Pages” folder, but more tricky to merge server side code, so we have to start being more careful about source code control. You may also find that you have to modify some links (like styles or JS files or web APIs) for going up or down a folder. Remember that
   a normal relative link like “mycode.js” means in the same folder (as the html file that’s running it)
   “../subdirectory/mycode.js” means up one folder then down into “subdirectory” to find the file mycode.js.

5. **Blog.**
   Add an entry to your blog page. As usual, tell what was easy, what was hard, what you learned (and what did you think were the most important new concepts this week). As mentioned in the requirements at the top of this document, provide 4 links: web_user API, web_user HTML page, Other API, other HTML page. Test that they all work.

6. **Publish.**
   PLEASE PLEASE follow the publishing instructions very carefully. If you are sloppy with where you place your files, you can get yourself really stuck and I’ll have to help you get unstuck. These are some common errors:
   a. Student publish their “.java” (source) files instead of the “.class” (compiled) files.
   b. Students forget to do the copy/delete/wait/paste the web.xml file after they have copied up their “.class” files. When you publish, you copy up web.xml, not sunweb.xml. There’s nothing much in the web.xml file, but it contains settings for the tomcat JSP application server and it lets tomcat know that it should go out and check for newly published classes. Otherwise, tomcat (being geared towards production, not development) assumes that no changes have occurred to class files (it is uses a copy of what you published previously).
   c. If you ever find that you are unable to delete any files under your web root folder, email me right away, I forward your email to our CIS department IT administrator and he resets your privileges. For this, you could even email him directly: stauffer@temple.edu. Be sure to explain your problem well. He’ll need to know your NetAccess user name and the folders/files you cannot delete on cis-linux2.
Example Deductions

-9: code not published.
-9: code too similar to a classmate’s.
-5: sloppy naming (names inappropriate, left the same as sample code) – server side code, client side code.
-3: unprofessional layout, layout that does not blend with home/blog page and/or no layout.
-2: data not appropriately formatted and/or aligned (e.g., dollar amounts, dates, integer).
-2: table too wide due to data not being allocated appropriately. You may need to combine database columns into one HTML column where appropriate (e.g., if you had city, state, and zip as three database columns, you might concatenate them into one HTML column. You might create double line cells like first name <br/> last name.
-2: image data columns not being represented by <img> tags and/or images too wide for page.
-2: code not indented (use NetBeans menu option "Source - Format").
+1: implemented click sortable columns in both HTML Pages (0.5 per HTML page).