Data Display Homework

Overview: In this lab, you will write AJAX calls (in JavaScript) to make asynchronous http requests to the Web APIs which you wrote in your previous homework. Asynchronous means non-blocking, in other words, the User Interface does not lock up waiting for the response. After receiving the response (in JSON format), the browser shall invoke your callback function which shall display the tabular database data on the screen, with a few client side bells and whistles - like click-sortable column headings and a textbox that filters the data. Sample code shall be provided that shows a “plain vanilla” HTML table (of web_user data joined with user_role data) and there will be other code that shows how to implement click sortable column headings and filtering. You shall write similar code that lists data from your “other” table joined with web_user.
Functional Requirements:

1. When the user clicks on the “Search – Users” link from your nav bar, web_user data (joined with role data) shall be displayed.
   - You may substitute some other text for “Users” in the nav bar, e.g., “Customers” or “SkateBoarders” or whatever works best with the web application you have envisioned.
   - This display shall show all the fields of all the records of your web_user database table joined with user_role, initially ordered by some field other than web_user_id (the order provided by the Web API).
   - The columns shall be presented in an order that makes the page useful and appealing, but the (initial) sort order shall be the first column in the HTML table.
   - You may combine some fields into a single column or leave all fields separate – try to make your page look good.
   - Be sure that your web_user table has at least one record with all fields populated and at least one record with all nullable non-character fields set to null. Test that these null values do not break your web application.
   - NOTE: As mentioned in the previous homework, if there are database design errors when you run the sample code, change your data design (instead of modifying the code).

2. When the user clicks on the “Search – [Other]”, data from your “other” table (joined with data from web_user) shall be displayed.
   - Substitute the name of your “other” table under the Search menu in your nav bar.
   - This display shall include all the fields from your “other” table, plus at least one non-key field from web_user (e.g., the email_address), initially ordered by some field other than the id of the “other” table. The initial sort order field shall be the first column in the HTML table.
   - The columns shall be presented in an order that makes the page useful and appealing, but the (initial) sort order shall be the first column in the HTML table.
   - Combine fields into a single column, where appropriate, so that the HTML table is not unnecessarily wide. For example, if you had fields address, city, cust_state, and zip, you could combine those fields into a single cell of the HTML table inserting a “<br/>” (new line) tag before city + cust_state + zip.
   - Be sure that your “other” table has at least one record with all fields populated and at least one record with all nullable non-character fields set to null. Test that these null values do not break your web application.
   - Your "other" table was required to have an image URL field. Your app shall build the URL into an image tag like this:
     
     \[
     <td> <img src='[imageURL]' /> </td>
     \]

     Your style sheet shall include rules to set your images to a small width so that your HTML table does not look bad. Make sure that the URLs in your data do not link to images with have huge file sizes since this will result in very slow rendering of the page.
3. In both displays mentioned previously, the data shall be displayed in an HTML table.
   o Each column in the HTML table shall provide a “click sortable” column heading.
   o Data shall be formatted and aligned appropriately, e.g., numeric data right aligned, dollar amounts must be formatted with $ and , separators.
   o If any database fields represent URLs, the URLs shall be incorporated into links like this:
     <td> <a href='[teamWebSiteURL]'> [teamName] </a> </td>
   o Data fields shall be combined into one cell where appropriate, to avoid overly wide HTML tables.

4. The HTML tables shall be linked to a “search filter” textbox that controls which rows are displayed.

5. Tables in your database were required to have certain nullable non-character fields (like date, int, decimal). This is because “real business applications” often have nullable non character fields which can make the code tricky to implement. Each table shall have at least one record with all values populated and at least one record where all null-able columns have null for their value. If your data does not have this, modify your data and test that your web application still works OK to avoid HW deductions. If you run your result set columns through the FormatUtils methods, things should work OK.

6. For any database error that could occur despite bug free code, your web application shall provide "user friendly" db error messages followed by the full technical error message, like this:

   "Database unavailable - please try later or contact your administrator. Database error message is: ..."

   It is important to show the actual technical error message (after the user friendly one) so that application support can resolve problems experienced when code is in production. You can test for this by not tunneling in when developing from home and trying to run a page.

7. Your blog page (as for every homework) shall have a blog that describes the work you did this week (what was hard, what was easy, and what important concepts you learned).
Design Specifications:

1. Each of the new “data display” links shall invoke some JavaScript that makes an AJAX call to the appropriate Web API and then injects the HTML table into the content area of your index page. You have been provided with two separate code examples – one that creates a “plain vanilla” HTML table.

2. This homework shall (like the Routing Homework) reuse HTML/CSS user interface by employing JavaScript components that inject content into the content area, as opposed to using “regular page links” where pages are copy/pasted with modified content.

3. This is how your project shall be organized (see image below):
   - The web root folder shall provide folders for icons, javascript code, pictures, style, and Web APIs.
   - Your project shall contain a dbUtils package with classes DbConn and FormatUtils inside. You can use the sample code as is for these two classes (but change the DB connection string to reference your credentials).
   - Your project shall contain a package named “model.webUser” that holds a StringData class and a StringDataList class. Use the sample code as provided (modify your DB design, if necessary, so things work).
   - Your project shall contain an additional packages named model.xxx (where xxx is the name of your “other” database table ). This new package shall have a class named StringData and StringDataList both modeled after the classes in model.webUser, but with all properties named appropriately.
     - Consistent naming helps organize your code, simplifies the coding process, and prevents errors.
Your web APIs shall have the least amount of code in them (getting a database connection, setting up the StringDataList and calling a class in your view package to populate the StringDataList from the database). It is good coding style to move code out of a JSP page (wherever possible) and down to the more reusable java class code. So that you understand what NEEDs to be in the JSP Page, it is these things:

- input (not in this HW, but in future),
- output (the out.print statement),
- database connections (as discussed in the Web API tutorial document) and
- anything related to the JSP session object (e.g., next homework when we work on Log On).

4. In addition, your web application shall conform to all the (“Good Coding”) requirements listed in the section entitled “Requirements for All Homeworks and Project” at the top of the 3308 web page. Please check that all links in your web application work properly while the URL never changes (since we are using “Single Page Application” design) - your look and feel shall stay the same regardless of what link the user clicks. All text shall be legible at all times. All text must be at least a little bit away from any visible edges (use margin and/or padding).

Homework Submission:

- Publish and test your web application. Remember to add your blog text for this week and test whatever your blog links to (before and after publishing, use relative links not absolute).
- Attach (into the Canvas assignment), the document described above, along with a zip file of your complete web application project folder.

Suggested Approach:

- Do the lab activity if you have not already done it. This will expose you to some of the code in 04_ClickSortableTables.zip which shows
  - how to create DOM objects and attach them to divs on the page.
  - how to make an AJAX call, supplying a callback function in which you convert JSON into a list of objects, then call an encapsulated function that turns the list of objects into a nicely formatted HTML table which you can then display on the page.

04_ClickSortableTables.zip has more sample code that is relevant to this homework, so study all the folders including the sort samples, parameter object, and search filter.

To simplify things, this sample code only used fixed JSON files (not live Web APIs generated by server side java/JSP). That is why you did not have to use the enhanced instructions (for installing/debugging/publishing) and why you did not have to tunnel in when developing.

- Next, test your Web APIs from last homework and be sure they still run. You basically replace “json/allWebUsers.json” (when you make the AJAX call) with “webAPIs/listUsersAPI.jsp”. Once you start using the live server side code, you’ll have to be tunneled in whenever you are developing, and you must use the enhanced installing/debugging/publishing instructions.

- In your AJAX calls, use relative URLs to reference your Web APIs, not fully qualified URLs. This is for two reasons: (1) so you can see server side messages that might be written by java/JSP code in your glassfish log and (2) so that you do not get the CORS error – Cross Origin Resource Sharing error (a security measure that tries to prevent a page from one page making an AJAX call to a different domain).
In addition to sample code 04_ClickSortableTables.zip, you also have 3308_04_dataDisplayV2.zip which just shows the building of a rudimentary HTML table but from a live Web API (whereas the other zip file shows making fancier HTML table from static JSON files).

Example Deductions

- -9 if no zip file submitted into Canvas (by the due date).
- -9 if no pages published by the due date.
- Up to -2: missing or incomplete blog content.
- Up to -4 if web user records not displayed or incomplete according to requirements.
  - -1: your web_user table does not include a record with all nullable fields null or there is such a record but it causes errors.
  - -1: unrealistic looking web_user data.
  - -2: if not clicksortable
  - -1: if not filterable
- -7 if other records not displayed or incomplete according to requirements.
  - -2: your “other” table data does not include a record with all nullable fields null or the data does include such a record but this causes errors.
  - -3: data from your “other” table was not joined with data from web_user and/or the page does not display at least one non-key field from web_user.
  - -2: your other table is lacking a picture URL field and/or this field is not populated with valid fully qualified URLs of images from the internet.
  - -1: the page renders too slowly because your test data references huge image files.
  - -1 the layout does not look professional. For example, you do not need to show the FK id to web_user. For example, you can combine more than one field into one <td> (table data) so that the HTML table is not so wide. For example, you have not styled the image to be narrow to fit nicely on the page.
- Up to -2: data in HTML tables are not formatted and/or not aligned properly (numeric should be right justified).
- Up to -2: lack of professionalism/originality. The page should look at least as nice as the sample provided for the Home Page HW (no text too close to visible edges, all text legible, use of subtle color scheme, fluid layout, no unwanted wrapping at any browser width, no large images slowing down page load). Original design, not overly similar to sample code nor classmate code.
- Up to -3: JS code not used (to inject content into the content area, for four links: home, blog, search – users, search – other.
- Up to -2 for what we notice when we “View Source”:
  - HTML/CSS/JS syntax errors which would show in red font from Firefox View Source.
  - poor coding style. Your code should be neat, properly indented, well named, and with no unnecessary code.
- Up to -2 for code organization (not meeting design specs). Students show their project opened up in NetBeans during lab activity.