Data Display Homework

Overview: In this lab, you will learn how to write database access Web APIs (in java/JSP) and then call these Web APIs from the browser (in JavaScript/ajax). You will be provided sample code that lists your users and you will write code that lists data from your “other” table as well as your associative table (joined with users and other).

Word Document Requirements: Read the Tutorial entitled “Using Java/JSP to Write a Web API”. Download, install, run, and study its sample code. A link to the tutorial and to the associated code is provided in Display Data HW section of the 3308 Web page.

After running and studying this code, generate each of the following error messages (by modifying the code or other means):

1. Missing Database Driver
2. Database Unreachable
3. Database Not Authorized
4. Syntax error in Sql Statement
5. Error Extracting Data from Result Set (bad column name)
6. Error Extracting Data from Result Set (wrong data type)

For each of the above types of errors, add the following information into your Word or wordpad document:

1. The type of error (as listed above).
2. Copy/paste the exact error message you got (either from the JSP page or from the GlassFish log).
3. Describe how/where you would normally expect to fix the problem.

Blog Requirements: To your blog content,

- Add a blog heading and section entitled “Data Display”. In here, discuss this assignment – what was hard, what was easy, and what were the most important things you learned. From this blog entry, be sure to link to the word document where you discuss data access errors (described just above).
- How to link to documents: Just sftp the document just like your other files (e.g., in your web root folder). Remember to use relative links - test that the link works before and after publishing.

Functional Requirements:

1. Server Side requirements. You are to write three Web APIs (named exactly as shown):
   - listUsersAPI.jsp: this Web API shall provide (in JSON format) an object that includes a possible database error, plus an array of objects, one object per user in your web_user database table. Each web_user object shall contain all the fields from your web_user table joined with your user_role table. The objects shall be ordered by web user id.
   - listOtherAPI.jsp: this Web API shall provide (in JSON format) an object that includes a possible database error, plus an array of objects, one object per user in your “other” database table (whatever you may have called that table and you were not allowed to call the table “other”). Each “other” object shall contain all the fields from your “other” table. The objects shall be ordered by whichever column you chose to select first in your result set (and show first in the HTML table in the UI).
o **listAssocAPI.jsp:** this Web API shall provide (in JSON format) an object that includes a possible database error, plus an array of objects, one object per record in your associative table (whatever you called it). Each associative object shall contain all the fields from your associative table as well as at least 1-2 non-key fields from each of the related records in the web_user table and your “other” table. The objects shall be ordered whichever column you chose to select first in your result set (and show first in the HTML table in the UI).

2. **Client side requirements.** When the user clicks on “Users” (or you may substitute link text that is more appropriate to your web app, such as “bakers” or “customers”), the user shall see (in the content area of the page):
   o All the records and all of the fields of your database table that is named web_user joined with user_role.
   o The columns shall be presented in an order that makes the page useful and appealing.
   o Records shall be ordered by whatever column you chose to put first (and possibly have a secondary sort on the second column, if appropriate).
   o NOTE: This code should have been provided for you as sample code. You should be able to just incorporate this code directly into your web application. If your web_user and/or your user_role table design does not match exactly what I prescribed in the HW#1 (data model/project proposal), then change your data design now (the earlier the better so it will be less work for you going forward).

3. **Client side requirements.** When the user clicks on a link that has text that matches the name you selected for “other” database table, the user shall see (in the content area of the page):
   o All records and all fields of your “other” table.
   o The columns and rows shall be presented in a useful and appealing order.
   o Records shall be ordered by the 1st and possibly 2nd column.

4. **Client side requirements.** When the user clicks on a link that is named like you named your associative database table, the user shall see (in the content area of the page):
   o All records of your associative table joined with your user table and your “other” table.
   o All fields of your associative table shall be shown (except its PK and FK fields) and at least one non-key field of your user and at least one non-key field of your other table. The columns shall be presented in a useful and appealing order (sorted by 1st and possibly 2nd column). Combine fields where appropriate.
   o Note: Since there is a pretty nice JS method called buildTable that you can use, you might want to “doctor up” the array of objects before passing them into buildTable. For example, you could iterate over the array, deleting object properties that you do not want to display and/or you could combine fields nicely into a single property. Here is a link regarding how to delete a property in JS:

5. **Nav Bar.** The sample code demonstrates how to do a drop down menu using JavaScript (don’t use older hover techniques since they do not translate to cell phones). Group the three above links together (for example under the word “Show” or “Data” or “Explore”) – this is so that your nav bar does not get too wide.
6. In all three user clicks mentioned above, the data shall be displayed in an **HTML table**.
   - Each column in the HTML table shall provide a column heading.
   - Data shall be formatted and aligned appropriately, e.g., numeric data right aligned, dollar amounts must be formatted with $ and , separators.
   - If any of database fields represent URLs, incorporate these into links. For example, suppose your table has one column that holds a teamName and another column that holds a teamWebSiteURL, the HTML that you create for one of the columns might look like this:
     
     ```html
     <td> <a href='[teamWebSiteURL]'> [teamName] </a> </td>
     ```
   - If any of your database fields represent images, incorporate these into image tags. You’ll probably want to include a width attribute so that you don’t get really large images in your tabular HTML table. Try not to link to images that have huge file sizes since this will result in very slow rendering of the page.
     
     ```html
     <td> <img src='[imageURL]' width='200' /> </td>
     ```
   - The data shall be combined into one cell where appropriate.
     - Note: it is likely that the associative data display will be too wide for the browser if you display each DB field in its own HTML table column. Think about how you want to display the data – don’t just have a one to one correspondence between fields from the JSON data and columns in the HTML table. For example, you might have a single column labeled “Address” that has [[street address, then a new line, then city, state, and zip on a second line]]. For example, if one of your db fields is a title and another an explanation, combine both into a single cell in your HTML table with the title inside of a header tag (e.g., `<h4>`) and the explanation inside of a paragraph `<p>` tag. Use creativity and make it look nice.

**Note:** you can tweak your data model this week, as long as you continue to meet all the requirements that are listed in the top link under the “Requirements for all HWs and Project” from the 3308 web page. However, tweaking the design in future semesters will require that you rework the old HWs so I don’t suggest it.

7. Tables in your database were required to have certain **null-able non-character fields** (like date, int, decimal). Each table was to 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 did not have this, take time to fix this now (or you will get points deducted from your HW). Your code has to be able to gracefully handle all of these data types, including null values. If you run your result set columns through the FormatUtils methods, things should be OK.

8. Make sure to always have "**user friendly**" **db error messages** followed by the full technical error message. For now, this just means that your code should test any database exception message and precede the message with a user friendly one. Example: when you attempt to open a database connection, it may fail or not. If it does fail, be sure to intercept the technical error message (that’s provided by the database) with a user friendly message like "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 people can resolve problems experienced when code is in production.

9. 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, what important concepts did you learn).
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.

2. This is how your project shall be organized (see image below):
   - The web root folder shall contain a directory for html partials, for javascript code, for images, for style, and for 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).

   ![Project Structure Image]

   - Your project shall contain a package named “view” which has a class named WebUserView and two additional classes named as you named your “other” and associative table.
   - Your project shall contain two additional packages named model.xxx and model.yyy (where xxx is the name of your “other” database table and yyy is the name of your associative database table). Both of these new packages shall each have a class named StringData and StringDataList.
     - The StringData class should be modeled after the Web User StringData class – one public String property per field in your database table (regardless of data type in the database table). The StringData class may also have fields from joined tables. For example, user_role fields were added to the webUser StringData class. The purpose of StringData for your web app is to hold pre-validated data (might not be a valid date, for example) and/or formatted data (dollar amount might hold “$” and commas, etc).
     - The StringDataList class is what we will use to hold a result set that we pull from the database. It needs a possible database error message and an array of (formatted) StringData objects extracted from the DB.
Consistent naming helps organize your code, simplifies the coding process, and prevents errors.

- All three of your WebAPIs (one provided as sample, two that you write) shall be stored in the same folder “webAPIs” under the web root folder. 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 away from 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).

3. 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:
- If you have not completed the previous homework (home page), you’ll need to get that done first.
- In the 3308 web page under this homework “Data Display”, you’ll see a link to various documents that provide new ways to install sample code, tunnel in (whenever you test a database access page from home), and publish. This is necessary because now we have more rich projects that include server side Java/JSP code and database access). Be sure you are using these updated/enhanced “How To” documents.
- If you attended the lab activity, you would have done this in lab, but if not, do it on your own:
  - In the 3308 web page under this homework “Data Display”, you’ll see a link to a tutorial that talks about creating Web APIs. Read and study that tutorial along with its associated sample code. Redirect the sample code so that it pulls data from your database (by changing the database connection credentials in dbUtils/DbConn). Back up the project you had from the previous homework, add the JAR files, add the packages from the sample code, and try to write a Web API for your Other Table.
- In the 3308 web page under this homework “Data Display”, you’ll see a link to some sample code that shows the client side JavaScript code that you need to make AJAX calls to your Web APIs. Install and study this code. Redirect this code to pull from your database. Modify the index page so it looks like your home page. Get the index page to show the data from your web_user table with your home page design. Move this code into your web application code and keep working on it, adding the link to your other table, then to your associative table.
Example Deductions

- 9 if no zip file submitted into Canvas (by the due date).
- 9 if no pages published by the due date.
- 2.5 if your blog content does not link to your error message document.
  - Up to -2 for incomplete error message document.
- 1 if you don’t have a blog that links to a data model image.
- Up to -1 for lack of blog content (where you tell what was hard/easy/important about this week’s HW).
- 0.25 if your name is not in the footer.
- 1 if you have not used a drop down menu for the three links added this week.
- 2 if web user records not displayed.
  - 1 if there are no web user records that demonstrate that your code gracefully handles null non character fields (membership fee and birthday).
- 3 if other records not displayed.
  - 1 if your other data does not have a null date or numeric (integer or decimal).
- 4 if assoc records not displayed.
  - 1 if your associative data does not have a null date or numeric (integer or decimal).
  - The select statement needed to join the associative table with the user table and the “other” database table.
    - 1 if the HTML table (that shows associative data in the content area) does not contain the descriptive field from your user table.
    - 1 if the HTML table (that shows associative data in the content area) does not contain the descriptive field from your other table.
  - 1 if the layout does not look professional. For example, you do not need to show the FK ids to user and other.
    For example, you can combine more than one field into one <td> (table data) so that the HTML table is not so wide.
- Up to -2 if data in HTML tables are not formatted and/or not aligned properly (numeric should be right justified).
- Up to -2 for 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). Also, your pages cannot overly similar to sample code nor classmate code.
- Up to -3 if JS/AJAX code not reused (to invoke the home content, the blog content and the three new content areas: users, assoc, and 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.