Web API Homework

Overview

In this homework, you will write a Web API, server side (Java/JSP) code that selects data from your database. To test that your Web API works (and after tunneling in), you run the JSP page directly from NetBeans and you should see raw data output onto a web page. If you have a JSON viewer plugin installed to your browser, you’ll see that data nicely formatted. You will be given a sample Web API that lists data from your users table (joined with the user_role table) but you will write your own Web API that lists data from your “other” table (joined with the user table).

After getting both Web APIs to work from your database, you will modify the code you did for the Display Data homework – instead of pulling “hard coded” data from a JSON file, you’ll pull live data from your Web APIs (for users and for your “other” database table).

Enhanced Instructions

In this homework, you will begin adding server side code to your web site. So...

- Test your NetBeans bundle installation to be sure you can run JSP page (add a JSP page, any name, accept all defaults, right click and run, see if it renders “Hello World”).
- Make sure to use the Enhanced instructions for installing sample code, for running pages on your PC/MAC (you have to tunnel in), and for publishing. You can find links for all of these enhanced instructions under this homework on the class web page.

Assigned Reading

Read the Tutorial entitled “Using Java/JSP to Write a Web API”. Download, install, run, and study its sample code. You’ll find a link to this tutorial and associated sample code under this week’s homework in the class Web page.

Word Document Requirements

After running and studying the sample code, copy/paste the following list of errors into a word document.

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:

1. Generate the error and describe (in the word document) how and where you generated the error. It follows that to fix this type of error, you would just have to undo whatever you did. Of course, the point of this exercise is for you to know how to fix these types of errors.
2. Copy/paste (into the word document) the exact error message you got - either from the JSP page that you see in the browser or from the GlassFish log.

Note: Students who have done the above exercise are much better able to find and debug their own errors when writing server side web code. This is why I ask you to do it 😊
Functional Requirements

You shall add the following Web APIs to your web application (named and organized within your project as shown):

1. **webAPIs/listUsersAPI.jsp**: This is basically provided as sample code but you must incorporate this code into your own Web Application, have it pull web_user data from your database. For things to work, your database design must exactly matches the requirements. If you get an error trying to run the sample code, do not change the sample code. Instead, change your database design to match the requirements. Otherwise all the sample code, all semester, will never work for you. **Your listUsers API shall provide (in JSON format)** a single object that includes
   - a database error message (such as connection error or syntax error within the SQL select statement), or empty string if there was no database error - AND -
   - an array of objects (one object per user in your web_user database table) ordered by web user id.
   - Each object in the array shall contain all the fields from your web_user table joined with all the fields of your user_role table (the join field, user_role_id, shall not be listed twice), plus an error message field that communicates any record level error (such as trying to convert a character string to a date type field) – if there is no record level error, that field shall just contain empty string.

2. **webAPIs/listOtherAPI.jsp**: this Web API (that you shall write) shall provide (in JSON format) a single object that includes
   - a possible database error message (like above) - AND -
   - an array of objects (one per record in your “other” database table - whatever you named that table and you were not allowed to name it “other”) joined with web_user, ordered by the primary key of your “other” database table (it helps you debug to know the order that the JSON data will be displayed).
   - Each object in the array shall contain all the fields from your “other” table and at least 2-3 fields from the joined web_user table (the join field, web_user_id shall not be listed twice), plus an error message field to hold any record level errors.

3. Tables in your database were required to have certain** null-able non-character fields** (like date, int, decimal). Remember that nullable means it is OK for the user to NOT provide a value and the database will still accept the record. Each table was to have at least one record with all values populated and at least one record where all null-able columns are null. If your data does 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 because methods in the FormatUtils class check for null and substitute empty string which makes for nice display on the page and also prevents “null value” exceptions from tripping up your code.
4. Make sure to always have "user friendly" db error messages for any error that might happen even when all code is fully debugged. For example, the database might be down one day and this is outside the control of your web application. Follow the user friendly message with the full technical error message that would enable a support person to understand and fix the problem. Example: if your code is not able to open a database connection, the error message should start out user friendly then get technical like this:

    Database unavailable - please try later or contact your administrator. Error: [technical message from the DBMS].

5. When the user clicks on the links under the search icon, the components shall be built from live data feeds from your database.

    This means that in your users.display JS function (and your “other” display function), you’ll replace the JSON file name in your AJAX call with the Web API reference. To be clear, instead of this:

    ajax("json/cars.json", processData, "listHere"); // processData is the name of the success function

    you’ll have this:

    ajax("webAPIs/listUsersAPI.jsp", processData, "listHere");

    Also, remember that the JSON file was just an array of objects, but what webAPIs/listUsersAPI.jsp outputs is more than that:

    {
    "dbError": "",
    "webUserList": [
      {
        "webUserId": "1",
        "userEmail": "sallyk",
        "userPassword": "p",
        "image": "http://cis-linux2.temple.edu/~sallyk/pics/s_k_2017.jpg",
        "birthday": "12/27/1991",
        "membershipFee": "$123.40",
        "userRoleID": "1",
        "userRoleType": "Admin",
        "errorMsg": ""
      },
      {
        "webUserId": "2",
        "userEmail": "johnd",
        "userPassword": "pass",
        "image": "http://example.com/johnd.png",
        "birthday": "10/15/1985",
        "membershipFee": "$99.99",
        "userRoleID": "2",
        "userRoleType": "Member",
        "errorMsg": ""'
      }
    ]

    So the start of your AJAX success function should first test for DB error, then pull off the array of objects (if there was no DB Error):

    ajax("webAPIs/listUsersAPI.jsp", processData, "listHere");

    function processData (obj) {
      if (obj.dbError.length > 0) {
        document.getElementById("listHere").innerHTML = obj.dbError;
        return;
      }
      list = obj.webUserList;
      // turn this list of objects into click sort-able filterable HTML table (component)
6. Leave your JSON files (from last week’s HW) in your project. So that they do not become totally unused, please add links from your HW3 (data display) blog to your two JSON files.

7. Like your previous homeworks, your web app will have drop down menus.

8. Your blog page for this homework shall have an entry that describes the work you did this week. From this blog entry:
   - Describe any server side database access code you may have written (could be PHP or dot net or python, whatever).
   - Discuss what important concepts you learned this week. Tell what you found easy and what you found hard or confusing about this assignment.
   - Link to the word document with the database errors (described in this document). To link to documents, just place the document into the “Web Pages” folder (in NetBeans) and sftp the document up to the web server as usual when you publish. Use relative links so that they work locally and after publishing. This link might look something like this (depending on how you named your doc):
     
     Click <a href="HW5_db_errors.docx">here</a> to see my document about java DB access errors
   - Provide relative links to invoke the two Web APIs (users and other). Test these links making sure they work locally and after publishing. Here is an example of how that kind of link would look:
     
     Click <a href="webAPIs/listUsersAPI.jsp">here</a> for the Web API that lists the users from my DB
Project Organization and Design Specifications

1. Like your first two homeworks, your web app shall use the provider style JS code for drop down menus and routing (injects components into the content area).

2. Under “Web Pages” your app shall have a “webAPIs” folder that holds the your two Web APIs (JSP pages).

3. Your Web APIs shall have the least amount of code possible, e.g., getting a database connection, calling a method from the view package to get a java object (an array of records from the database), then “out.printing” the JSON of that to the web page.

4. Your Web APIs shall reference reusable java code that is organized, as shown, under “Source Packages” below.

5. Your app shall have a dbUtils package that includes DbConn (a wrapper class for java.sql.Connection) and FormatUtils (a class full of static methods that format database fields according to data type). The only change you need to make in this package modifying your database credentials in DbConn.

6. In addition to the code shown under Source Packages, the app shall have a folder/package named model.xxx (where xxx is replaced with the name of your “other” database table). In this “model.xxx” package, shall be two classes named StringData and StringDataList (naming these classes like model.webUser makes the overall code easier to understand and maintain).

9. In addition, your web application shall conform to all the (“Good Coding”) requirements listed in the section entitled “Requirements for All Homeworks and Project” in the class web page, especially good naming, which goes a long way to providing well documented, understandable code. Each week, you are enhancing your web application, so requirements from previous homeworks should also be met.

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.
Example Deductions

- 9 if no zip file submitted into Canvas (by the due date).
- 9 if no pages published by the due date.

Blog:
- -1: for not describing what was easy/hard/important about this week’s homework.
- Up to -3: if blog does not link to your error message document and/or document is incorrect or incomplete.
- -2: there are no links to run your Web APIs and you did not name the APIs as prescribed in the HW, so we cannot easily find your Web APIs to test them.
  - -1: if you did not link to your Web APIs but you named them as prescribed so we could easily type in the names and test your work.

Web User API
- -2: User Web API has error messages or does not exist.
- Up to -1: Your User Web API either does not list enough records or the records listed are not realistic looking.
- Up to -1: You did not have at least one web user record with all fields populated OR your data did not have at least one record with null in the nullable fields (membership fee and birthday).

“Other” API
- Up to -6: Other Web API has error messages or does not exist.
- -3: you did not include web_user fields in the output from your “other” Web API.
- Up to -2: This Web API either does not list enough records or the records listed are not realistic looking.
- Up to -2: You did not have at least one record with all fields populated OR your data did not have at least one record with null in the nullable fields (check the database homework for requirements regarding nullable non character fields).
- Up to -3 if your web site still looks unprofessional (problems with HTML/CSS and/or content on the home page does not match with the database you designed) and/or lacks originality.
- Up to -3 if routing not employed (to invoke the home content, the blog content).
- 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 to the TA during lab.