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, you can run it 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. Alternatively (and this is how you will test your published Web API), you can type the URL of the Web API directly into a browser. You will be given a sample Web API that lists data from your users table (joined with the user_role table). You will write your own Web API that lists data from your "other" table (joined with the user table).

Word Document Requirements

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.

After running and studying this 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 did that. It follows that to fix this type of error, you would just have to undo whatever you did.
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 are to incorporate it into your own Web Application, have it pull data from your database, and make sure that your database design exactly matches the requirements (or else the sample code will not work for you). 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") ordered by the primary key of your "other" database table. (You never want to present an unordered list of data.)
    - Each object in the array shall contain all the fields from your "other" table joined with all the fields from your 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 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 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 provided by the DBMS].

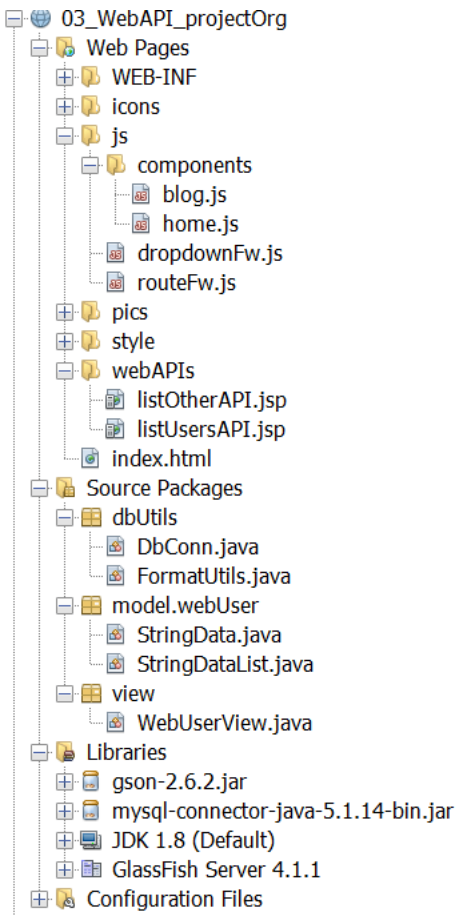
5. Your **blog page** (as for every homework) shall have a blog entry that describes the work you did this week (what was hard, what was easy, what important concepts you learned). From this blog entry:
    - Describe any server side database access code you may have written (could be PHP or dot net 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 here to see my document that shows how to create and fix database 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 here to see my Web API that lists all the users in my database.

Design Specifications:

1. This is how your project shall be organized (see image below):
   - You'll begin with the project organization from the previous homework, which means you'll have a **js** folder with components in it (like the ones that inject home content and blog content) as well as the JavaScript code for dropdown menus and routing.
   - You'll put the JSP pages under a folder named webAPIs.
   - You'll 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 is to change the database connection string (in DbConn) to be your credentials.
   - You'll have a model.webUser package with classes StringData and StringDataList inside, as provided from your sample code.
   - You'll have a model.xxx (named the same as you decided to name your "other" database table – you can't name this package model.other). This package should have classes StringData and StringDataList modified to be appropriate for whatever fields you have in your "other" database table. As mentioned in functional requirements above, the model.xxx.StringData class must have all the fields from your "other" table plus all the fields joined from your user table (but with the join field, web_user_id, listed only once).
   - You'll have a view package with two classes inside, WebUserView.java (provided in sample code) and a second class (named appropriately for your "other" database table) that extracts data from your other database table, placing the result set into a model.xxx.StringDataList object.



- Both of your WebAPI JSP pages (one provided as sample, the one that you write) shall call java code to do as much work as possible (because java code is more reusable). So what shall remain in the JSP page is this: getting a database connection, declaring/initializing the StringDataList object, then calling a class in your view package to populate the StringDataList from the database. So that you understand what NEEDs to be in the JSP Page (or any server side page), it is these things:
    - input (not in this HW, but in future – and this input would be in the form of URL parameters, e.g., "?product=123&dept=furniture" at the end of the URL),
    - output (the out.print statement that puts data into the page),
    - 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).

2. 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.

Suggested Approach:

- This homework presumes that you (by completing previous homeworks), you already have a web site with links for home and blog AND that you have a database with populated tables (user_role, web_user, and an "other" table that you chose to create).
- In the class web page under this homework, you'll see links 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 your web application is more sophisticated, including server side Java/JSP code that accesses a database). Be sure to use these updated/enhanced "How To" documents.
- The class web page includes a Web API tutorial and related sample code. Read and study this tutorial and the related code before starting to work in earnest on your homework.
- If you attended the lab activity, you would have gotten a good start in lab, but if not (or if you didn't have time to do the last extra credit part), you can do that on your own now:
  http://cis-linux2.temple.edu/~sallyk/cis3308/03_webAPIs/webAPI_activity_yr.html
- After getting a Web API that extracts just two fields from your "other" table, enhance your model.xxx.StringData class to include all the necessary fields and make the changes to your SQL SELECT statement and to the model.xxx.StringData constructor that extracts all the fields from the result set into the StringData properties.
- Test your Web APIs locally, then publish (using enhanced instructions) – REMEMBER that you MUST always copy/delete/paste the file web.xml AFTER updating any java classes on the server or else Tomcat will not recognize or use your newly published class files. Updating web.xml tells tomcat that your web app has been redeployed and so it needs to get new versions of your java class code.

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.
    - -1: if you did not link to your Web APIs but you named them as prescribed so we could type in the names and test your work.
    - -7: 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.
- **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 -5: 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.