Web API Lab

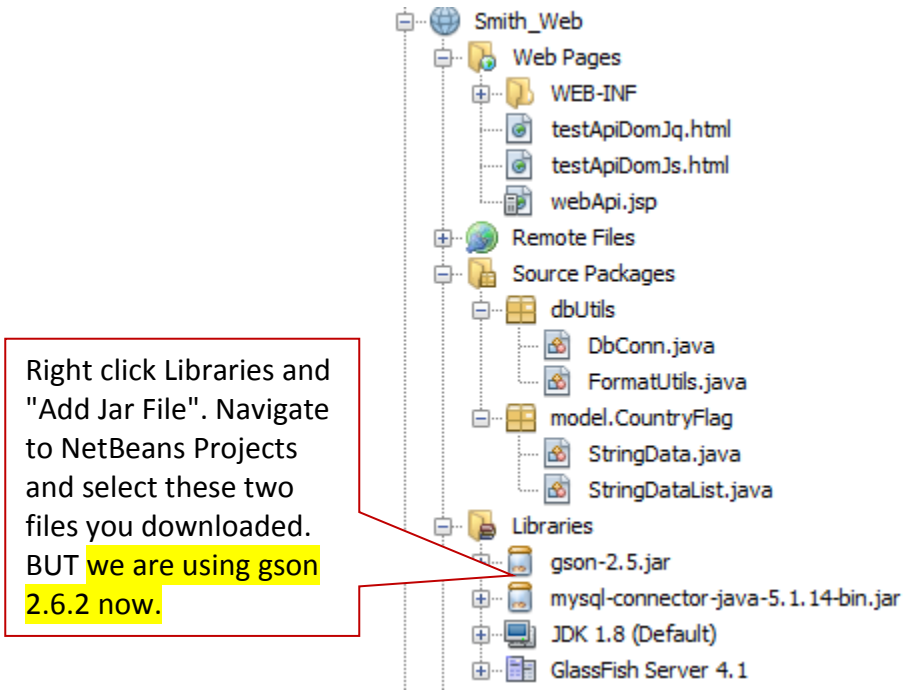
Overview

In this lab, you will produce three deliverables:

1. A server side Web API (named ***webApi.jsp***) that accepts an input parameter, queries ***your*** database, and then returns a JSON string that represents an array of objects (records) that match that input parameter. This web API will be written in java/JSP. This will be very much like the sample page: getCountryFlagsAPI.jsp
2. a (client side) html page (named ***testApiDomJs.html***) that invokes your Web API using the AJAX technique and javaScript (no jQuery) and modifies the Document Object Model, creating a HTML table and adding rows and columns of data into that, from the JSON string. This page will be very much like the sample page: testAPI_javaScript_DOM.html
3. a (client side) html page (named ***testApiDomJq.html***) that invokes your Web API using the AJAX technique and jQuery that works functionally the same as the page described just above.

You will add a "Web API" blog to your labs page and link to all three of the above files.

Project Organization



When you have finished this lab, you will merge this project with your current project so that the source code of your web site continues to be inside of a single local project.

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.

Server Side Functional Requirements

Your web site shall have a page named ***webApi.jsp***. This page shall

1. Extract the query parameter (from the Web API call).
2. Open a database connection object then pass the query parameter to the StringDataList constructor so it can query your database table (the table you created last week, not the user table, not the role table, I will refer to this table as ***"your 3rd database table"***). This query shall be a SQL select statement that utilizes the LIKE SQL keyword to find any and all records that match the given query parameter.
3. Close the database connection – this is important otherwise we'll have database connection leaks and this eventually brings down the server, depending on the load experienced by the server at any given time.
4. Convert the StringDataList object to JSON (using the GSON library).
5. Write the JSON (out.print) as a response to the web API call.

Server Side Design Specifications

Your web application (NetBeans project) shall:

- Include a ***dbUtils package*** and its classes from the sample code (DbConn, FormatUtils) with only this change: class DbConn shall have your database credentials instead of my database credentials (twice – once for the connection string for when you are tunneling in from home and the other for after your project is published). You can add extra functionality to FormatUtils if you need to.
- Include a ***model package*** with another package inside – this (inside) package shall be named the same as your 3rd database table and it shall contain any code that represents records from your 3rd database table and/or the list of records from your 3rd database table.
    - In the example on the next page, the name of my package is model.CountryFlag, which means it is a package named "CountryFlag" inside of a package named "model".

  In the ***model.yourDBTableName*** package you are to have two classes,
    - **class named StringData** (this is just a bundled object that has one property per field in your database – plus an extra field for "field level" error messages that will help you as you develop your code).
        - As an example, a field level error could be generated if you tried to extract a field name that does not exist in a result set or you tried to access a field which has a null value in the database.
        - All properties in the StringData class shall be strings, even if the corresponding field in your database table is not a string. StringData can represent pre-validated user input (in a future lab) and/or it can represent formatted data from your database (e.g., a dollar amount with commas and dollar sign).
        - If you leave all the data members public, you do not have to add the getters and setters and the syntax for users of this object is easy: ***obj.fieldName = "value"*** instead of ***obj.setFieldName("value")***
    - **class named StringDataList** which is a list of StringData objects plus a possible "list level" database error message (String). As an example, a list level database error could be database down, invalid sql statement – anything prior to trying to extract data from the result set.

Client Side Functional Requirements

testApiDomJs.html shall:

1. Provide a text box where the user can enter a value that shall be used to filter their data request. This value will be used as query parameter to the Web API JSP page that you just created.
2. Using just javaScript (no jQuery or other javaScript library), make an asynchronous ajax call to your Web API, passing the user input as a parameter along with the call.
3. Display the response of the Web API by adding an HTML table to the page and filling its rows and columns from the JSON data – in a useful and appealing way.
4. Have any layout that makes sense with the API that it calls – as long as the page looks professional. The layout can be the same as a previous lab or different. If your layout for this week is different from previous weeks, create a new external style sheet just for this week.
5. Provide instructions (text on the page) that let us know what we need to do to test/grade your functionality. Pre-fill a value in the text box that will yield some results (but not too many). These instructions should not make your page look less professional because any user would need to know how to use the page.

testApiDomJq.html shall:

1. Have the same functionality as testApiDomJs.html page except the code will use jQuery wherever possible.
2. Have the same layout and reference the same external style sheet as testApiDomJs.html.

Navigation Requirements

As mentioned in the overview, you will add a "Web API" blog to your labs page and link to all three of the deliverable files (webAPI.jsp, testApiDomJs.html, testApiDomJq.html). If you like, you can add a link or two into your real nav bar (that is included in index.html and labs.html using Angular ng-include), but you do not have to. If your nav bar is getting too full, use dropdowns (check lab 1 web design examples for help if you need it).

Program Style Requirements

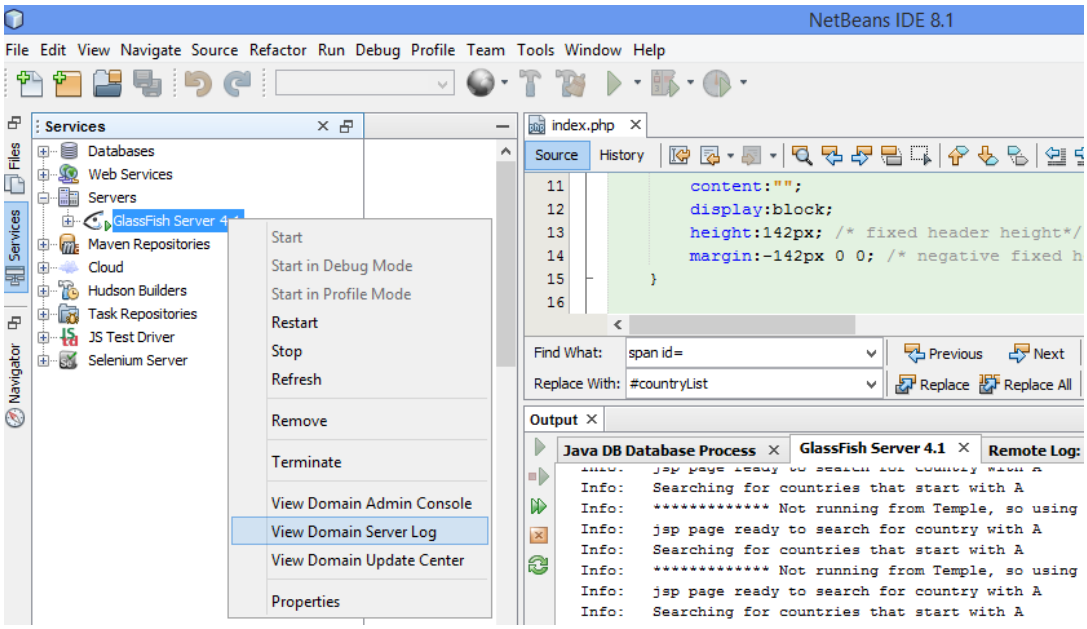
Check the "All Labs and Project Requirements" section of the 3344 labs page.

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 searchfor 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.



Submission

- Test locally (making sure everything still works from previous labs)
- Then publish and test what you published.
- Then submit a zip file into blackboard by the due date/time.

Suggested Approach

1. If you have done the Web API lab activity (that is described on the 3344 labs page, just prior to this Web API lab), you should have a sample NetBeans project that can access my database (but you must tunnel in if you are working on a non-Temple PC/MAC). If you have not done the Web API lab install the sample code (provided from this lab on the labs page), tunnel in if necessary, and run the JSP page locally (getCountryFlagsAPI.jsp).
2. Leave my sample project "as is" so thay my project always works, so you can use it for reference, as you debug your code.
3. Create a second project, and name it with your last name in it (this will become your new primary web app). Copy in my sample code (add the two jar files to the libraries folder, copy/paste the packages, copy/paste the HTML/JSP and any auxilliary files to the "Web Pages" Folder.) Rename getCountryFlagsAPI.jsp to be ***webApi.jsp.***
4. In your project, add a new package named "model.yourDbTableName" (substituting the name of your database table).
    a. Copy over the StringData and StringDataList classes from model.countryFlags into your new package.
    b. Change the field names in the new StringData (so that they match the field names of your database table) – remember all data members should be of type String even if the corresponding data type in the database is not string. This is because StringData will be used to hold pre-validated user input data and/or formatted output data.
    c. Change the code in StringDataList so that it reflects the SQL select statement that you need to get records from your table. Fix compiler errors (in StringDataList) that were created by your changing the names of the data members of StringData.
    d. Change the connection string in dbUtils.DbConn to be your credentials. You have to change it in two places because there are two connection strings – one for when you are running from home (tunnelled in) and one for when it runs on the web server after publishing (no tunneling used).
    e. Edit the webApi.jsp page so that it imports from your model.whatever package (not my model.countryFlag package). At this point, I think you should be able to run webApi.jsp, but only if you are tunneled in (if you are using a non-temple PC/MAC). When you run webAPI.jsp, If your JSON comes out ugly have not installed the JSON View Chrome plugin, do that so you can see nicely formatted JSON in your browser, instead of it being all jumbled together. JSON View actually provides JSON validation as well as formatting.
    f. Keep working until your JSON data shows no dbError at the top. If you have not already stumbled across a list level error in the dbError property (e.g., database unavailable, SQL syntax error), generate one on purpose just so that you understand a bit better how the code works.
    g. Check test for field level errors (would show as the last property of the record objects that are array elements). If you have not already stumbled across a field level error, you can force a "field level error" by trying to extract a column that does not exist in the result set (in class StringDataList) or by trying to extract data of the wrong type.
    h. Make sure that everything in the server side code is named well. Where I have named things generically (like recordList), you can leave those names alone, but do not leave any reference to names like CountryFlag (or anything non-generic) in your code.
    i. Test your web API by "URL Hacking" (type something like "webApi.jsp?q=A" at the end of the URL in your browser).

5. Next, begin to work on the client side.
    a. Try to get ***testAPI_javaScript.html*** to work. Note: this is not required for submission but it is a good exercise.
        - To make things easy on yourself, be sure that your javaSript property names (for your database fields) match exactly the java property names in your StringData object. This is because gson goes by the field names – if they match, you do not have to code up all the name mappings.
        - Make sure that you are ***Running*** your code (from NetBeans) not Viewing it – you need to see a URL that starts with "localhost" in the browser (or else the HTTP requests will not work).
        - Use "console.log()" profusely as you are debug your javaScript. You can also do something like this: ***console.log(object)*** – in the console area, you will see the object and you will be able to open/close it like a tree view. Make sure to always have the chrome debugger open (F12 from chrome).
    b. Rename testAPI_javaScript_DOM.html to be ***testApiDomJs.html.*** Note that this page is pretty similar to your goal for this week's lab. Use the same techniques (names, etc) as you used for the above page.
    c. Rename testAPI_jQuery.html to be named ***testApiDomJq.html.*** Note that you will have to do some internet research to learn how to create DOM elements like table, tr, td using jquery instead of javasript.
        - Using jQuery can be a little daunting, because an error can occur in "jQuery land" instead of your code. However, you should see a stack trace and be able to determine where (in your code) the problem actually started. Remember that you can do a console.log() even within your jQuery code—this is because you are actually writing javaScript which is just calling the jQuery javascript library.
6. After backing up your web application (e.g., that you had from previous labs) and after backing up your project that you just got to work, merge the two projects together (I guess you could copy either way). Make the layout of your two new html pages (testApi…html) have the same look and feel as your index and labs page.
7. Add a blog to your labs page and link to the two new testApi…html files, as well as to the webApi.jsp page (should just show raw JSON).
8. 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.