AngularJS Intro Homework

Overview

In a previous homeworks,

1. You created a database with two tables in it (your "User" table and your "Other" table).
2. You created a Web API (JSP page) that retrieved all the columns of all (or some) of the records from your "Other" database table and displayed them in JSON format. Then you wrote an HTML page that invoked your Web API using a technique called AJAX.

In this homework, you will

1. Create an HTML page that displays all the columns of all the records from your “Other” database table, but this time using the JavaScript Single Page Application Framework AngularJS (and its user sortable column feature) to invoke the Web API you wrote last week.
2. Create a second HTML page that will allow the user to insert data into your “Other” database table by calling an “Insert Other Web API” (JSP page) that you will write. Just to avoid mixing up too many new concepts, the insert page will not use AngularJS. The “Insert Other Web API” will:
   - Read all the user entered values from the URL parameters of the AJAX call.
   - Validate each value, and then (if everything validates OK), insert the record into the db table.
   - In either case (success or failure), the Web API write a response letting the client page about any errors validation errors and/or that the insert worked OK. (Even though the client side may also validate – we also want our server side code to check for errors and make error messages available.)

Database Requirements

Double check that your database design meets the requirements of the database lab.

- If your database design does not meet the requirements (for example, if it does not have a non-required non-character), you will lose points in many areas from your Angular homework grade.
- If you have used any SQL keywords as table or field names, it is very likely that your SQL commands (SELECT, INSERT, UPDATE, DELETE), as issued by your Java code, will not work properly.
- Your “other” table was NOT supposed to contain people. Your “user” table was for that.
- If necessary, modify your database design now (to avoid Angular homework deductions) and then make sure that your Web API lab still works (to avoid project deductions for that functionality).
- The database lab also had requirements for your data. For example, you were to have at least a certain number of realistic looking records in each table and your “other” table needed to contain valid image URLs. As you work through your Angular Homework, you will certainly be inserting, updating, and deleting records, but try to maintain the minimum number of realistic looking records at all times. You can do this, for example, by using MySqlWorkbench if you need to quickly insert a bunch of test records so that you can debug delete functionality.

Web API: API stands for Application Programming Interface (essentially a method call that has no user interface, a method that is exposed so that it can be called by external code). A Web API then is a server generated page (e.g., JSP page or PHP page or aspx page) that is only data (e.g., JSON format), has no HTML user interface.
Project Organization / Design Specifications

You will be producing a lot of files as you complete your Angular Homework and it is important to organize them properly. Plan to organize your files as shown below.

Put all your files for the Angular Intro Homework in a folder (under your web root folder) named "08-angular_intro" (only a few of the weekly homework folders are shown, not all).

Put your JavaScript files into a “js” folder under "08-angular_intro".

Put your Web APIs (JSP pages) into a “webAPIs” folder under "08-angular_intro".

Link to “08-angular_intro/index.html” from your labs page.

The java classes (that are referenced by your Web API JSP pages) go here, organized by packages (folders). Feel free to include the dbUtils package, as is, and use the basic java code that's in there.

This package should be named after your “Other” database table (not named “model.person” and not named “model.other”). It is fine to leave the classes inside named as they are (e.g., StringData.java, ...) – these generic names could be used for every model package that you might have – one per database table.

APIs that just “get” data from a database table should reference code from the “view” package (Model – View – Controller design philosophy).

Right click "Libraries" and add the mysql database driver JAR file and the gson JAR file that handles JSON ↔ POJO conversion (POJO is “Plain Old Java Object”).
Server Side Web API Requirements (JSP Pages):

1. **getOtherListAPI.jsp** *(copy/paste your webAPI.jsp from a previous lab)* shall return a JSON string that contains:
   - if no database error:
     - a database error message containing ""
     - an array of objects (one object per row of data in your Other table) where each object has all the columns in your Other table (plus a record level error message). *If any record has null in its non-character null-able field, convert this to "" (empty string) so you don't throw any null pointer exceptions.*
   - otherwise (database error):
     - the database error message
     - an empty array, e.g. [ ], no objects in the array.

Note: one easy way to test the database error case is to not be tunneled in when testing from home.

2. **insertOtherAPI.jsp** shall:
   - Accept a (JSON-ized) "StringData" object (which has one String property per column in your Other table plus a record level error property). This object represents the pre-validated “Other” record that is to be inserted into the database (that’s why the StringData properties are all Strings, even for fields that will be inserted into the database as other data types like date or integer or decimal).
   - Validate (server side validation) all the values from the JSON string and, if all are valid, attempts to insert the data as a record in your User table.
   - Return back to the page a (JSON-ized) StringData object (with each field holding the possible field level error messages) and a record level error message.
     - If the insert was successful, then all the properties of the StringData object shall contain "" (empty string), signifying that all fields passed validation. The record level message shall also contain "".
     - Otherwise, the StringData properties shall contain field level error messages (see examples below) and a record level error message.

**Examples of field level error messages:** “Required”, “Please enter an integer”, “Please enter a number”, “Please enter a valid date”, “Exceeds maximum (xx characters)” (for this case, and show them the first xx characters that they entered and allow them to modify their input to fit within the allowable number of characters).

**Examples of record level error messages:** the database exception message (as provided by MySql in your Java catch block), or “database unavailable”, "that email address already exists", or "you must be logged in to perform this operation" (you will implement this error message in Part III). To be clear, you are being asked to "intercept" known errors (like the tunneling error or duplicate value for unique field) and convert them to user friendly messages, but for other errors (unknown/unexpected error like sql syntax error), just show the actual database exception message.
Client Side Requirements (HTML, CSS, JavaScript):

1. As shown in the project organization diagram, your code for this week will have two main html files: index.html and insert.html. The nav bar of both pages shall have these links:

   ✓ The **Home Link** shall reference index.html, a file that invokes your "Get Other List Web API" and then displays all that data (all row, all columns) in a HTML table (like Angular tutorial example "05_get_with_repeat"). The HTML table shall provide Sort/Filter functionality (when the user clicks on the column heading) like Angular tutorial example "06_sorting_and_filtering". However, your page shall NOT list people – it shall list data from your “other” database table (that holds something other than people). Also, your HTML table shall have one column that shows images – place an `<img>` tag inside cells in one column and set the src attribute of the img tag to be the image URL that you have stored in your database. Set the width attribute of your img tags to be something like “150px” to control the look of your HTML table.

   ✓ The **Insert Link** shall reference insert.html, a page that implements “Insert Other” by calling the Insert Other Web API. This page shall provide labeled textboxes (one for every column of data in your user table) and a "Submit" button. Once the Submit button is clicked, the Insert Other Web API shall be invoked (passing a JSON-ized version of a user object with all the user inputs as attributes). The functioning of the API was discussed earlier in this document. When the page receives the AJAX response, the page shall display all field level error messages (which would each be “” empty string if the field passed validation), as well as a record level error message (e.g., success or a DB error or a validation message like “please try again”). Field level error messages shall be placed next to the textbox where the user error was made.

   ✓ The **Labs Link** shall link out of the Angular Homework SPA and back to your labs page:

   ```html
   <a href="../labs.html">Labs</a>
   ```

2. **Layout.**

   ✓ Your layout can be simple, but try to make it look somewhat professional and have the color scheme similar to your home page. If you decide to use background images, make sure that they do not interfere with the readability of your content.

   ✓ You can use ng-include statements if you like, for HTML code reuse within this folder, or you can just have both files have the same layout by copy/pasting HTML code and referencing a single external CSS file.

3. **Blog** in your Labs page that discusses your work, provides a link to Angular/index.html and direct links to each of the three Web APIs that you wrote this week (each link having an example URL that shows it’s working).
Submission:

- Test that the source code for ALL of your homeworks is working correctly inside of a single NetBeans project - this is called regression testing. Make sure your Angular Homework did not clobber java classes needed by your Web API lab. DO NOT have multiple projects locally that you publish/merge into one project on the web server.
- Check that your labs page has blog for this lab, including links to your Web APIs (with hard coded URL parameters that you can use for testing and we can use for grading).
- Publish and test what you published, then submit a zip file into blackboard by the due date/time.
- Remember that publishing is more involved now that we are using java classes. Refer to the publishing instructions provided in the Web API lab if you have trouble with publishing.
- Follow the "Requirements for All Labs and Projects" at the top of the 3344 labs page.

Suggested Approach:

1. **Lab Activity:** If you completed the first Angular Lab Activity, you should have gotten a good start with "http get and ng-repeat", "sorting and filtering" "routing for layout and with parameters". If not, study the Angular tutorial code (Sally’s site, Tutorials – Front End – AngularJS).

2. **Do one piece at a time.** Start working on getting the index page working (Web API should already be done). THEN work on the insert API followed by the insert html page. There are two separate sample zip files to help you with this.

3. **Install the sample code** on your development PC/MAC (in its own project) and study it. Experiment with a copy of it. Make sure to always have a working copy of the sample code installed on your PC/MAC.

4. If your **project is malformed** (you are getting weird compiler errors), create a new project and copy down the Source Packages and client side code (and add the two jar files to the Libraries folder). Carefully follow the instructions for installing a sample program from a zip file (you’ll find these instructions in the Web API lab). If this does not help, maybe your installation is bad. If so, uninstall glassfish and netbeans, then install the latest java JDK (even if you already have a JDK installed), then reinstall the NetBeans bundle, create a new web app project, add a JSP page to the web folder, run the newly added JSP page to see if your installation is OK.

5. When you are developing from home, you must be **tunneled in** so that your Web APIs (JSP pages) can access the DB.

6. Set Chrome as the **default browser** in NetBeans (Tools – Options, Default Browser pick list) and make sure you have installed the **JSON View Plugin** for Chrome.

7. **Write JSP Web APIs first** and run them directly from NetBeans (right click Run the JSP page). If it is a Web API that expects input parameters, use URL hacking to “hard code” the input parameters. Only when the Web API is working as expected, should you begin to work on the related client side functionality. Remember that the top line of a Web API JSP page needs a special **JSON content type** (this is different than a JSP page that would provide a user interface).
   - Use **System.out.println** to write debug messages while developing your Web APIs, from JSP and/or java (server side) code. You'll see these messages in the Glassfish Server Log (in the output area below the editing area). If you are a MAC user, click on Services (instead of Projects in upper left), open "servers", right click "glassfish server", select "view domain server log". If your code was published, you would not be able to access the tomcat server log where your messages would be going...

8. **Do not develop on published code** (YIKES!) This would be an unimaginable amount of work and very hard to debug (you wouldn’t have access to the glassfish server log to see debug messages). Run/test your project locally and only publish when things are working.
9. **Only write a few lines of code between testing.** Whenever you get something working well, **backup** and name the backup so that you will know what it is.

10. If you see a **database error in the JSON** of the web page,
   - check that you have added the mysql jar file to your project’s libraries (the error message would complain about mysql driver),
   - check that you are really tunneled in (the message would be something about connection),
   - check your username/password in your DbConn (message would be about invalid username/access)
   - It could show you a SQL syntax error for the delete statement, so you'd fix that in your java code.
   - You may have done a results.getObject("someColName") and tried to run that through a formatter that expects a different type. Make sure that you are formatting data according to the type that it is in the DB.

11. **HTML/JavaScript debugging.**
   - Put “debugger;” on any line of JavaScript code where you want the Chrome debugger to start, then you can single step through your code (see image of debugger on Sally’s JavaScript Tutorial).
   - Use `console.log` to write debug messages. If you console.log an object (by itself) you’ll see an object in the console log that you can open and discover its properties and values. You can see the console output if you press F12 from chrome and click on the console tab.

12. Remember that you must **"Run" (not "View")** the HTML pages because AJAX makes http requests which only work if there is a web server (localhost) communicating (http) with a browser. Ajax uses the request and response of the http protocol and that requires that a web server (apache) is involved serving up pages. Once you have Run a page, you can refresh to run again (after making changes). If you feel that you are **not running your latest changes:**
   - Right click and run index.html (instead of just refreshing whatever page you were testing).
   - Hold down the Control key while you refresh your browser (it is supposed to not use cached items)
   - From your Browser’s menu, clear your history/cache and/or Open an Incognito Window (from Chrome’s menu icon, it’s the third option).
   - Right click on your project and select "Clean and Build" (to recompile everything).

13. Name your **JavaScript property names exactly** the same as your **StringData property names** (in your server side, java/JSP code) – because of "automatic conversion tools" that we use.
   - The GSON class converts POJOs (java objects) \(\leftrightarrow\) JSON on the server side.
   - On the client side, JSON.stringify() and eval() / JSON.parse() converts javascript objects \(\leftrightarrow\) JSON.

14. If you are getting a **CORS error** (Cross Origin Resource Sharing), you are probably using a fully qualified URL (starts with "http://" instead of a relative URL in your ajax call. A CORS error would not occur unless a page from one domain (e.g., localhost) is requesting data from another domain (e.g., your published Web API).