AngularJS Intro Homework

Contents

1. Overview ................................................................. 2
2. Database Requirements ........................................... 2
3. Navigation Requirements .......................................... 3
4. Styling Requirements ............................................... 4
5. Project Organization Specs (For Routing) ..................... 4
6. Requirements for Click Sortable HTML Tables (with Client Side Filtering) ........................................ 5
7. Additional Styling Requirements ................................. 5
8. Overall Project Organization Requirements .................. 6
9. Blog Requirement ...................................................... 7
10. Submission ............................................................. 7
11. Suggested Approach ................................................. 8
12. Tips and Troubleshooting .......................................... 9
1. Overview

In previous homeworks,

1. You created a database with two tables in it (your "User" table and your "Other" table).
2. You created two Web APIs (JSP pages) - one that retrieved all the columns of all the records from your "User" table and another that retrieved all the columns of all the records from your "Other" database table. Both Web APIs displayed the data in JSON format. Then you created two HTML pages that (using JavaScript) made an AJAX call to invoke your Web APIs and display the data, nicely formatted on your page.

In this homework, you will create a Single Page Application (SPA) contained in folder “08_Angular” that invokes the Web APIs you wrote last week and also uses some of Angular features:

1. Routing, which lets you inject content into a layout like you did using “roll your own code” in the “Home Page” homework.
2. Click sortable HTML tables with client side filtering of data.

Note: If you have not written the Web APIs of the previous homework, you will only be able to complete the routing aspect of this homework, not the rest (click sortable HTML tables with client side filtering). To make your single page web site be self contained this week, copy/paste listOtherAPI.jsp and listUsersAPI.jsp from last week’s folder into this week’s folder. BUT under your “Source Packages” (java files), you only have one copy of everything – to be shared by all JSP pages (Web APIs).

2. Database Requirements

Before you start working on this assignment, double check that your database design (and test data) meets the requirements of the database homework.

- Note: Your “other” database table is NOT to contain people. You have a web_user table for that.
- If you have used any SQL keywords as table or field names, it is very likely that your SQL commands, as issued by your Java code, will not work properly. Google “SQL keywords” to be sure you are not using any of these as table or field names. For example, “state” is a field name you might select but it is a SQL keyword.
- If necessary, modify your database design now (to avoid this and future homework deductions) and then make sure that your Web API Homework still works (to avoid project deductions for that functionality).
- You were to have certain non-character nullable columns when you designed your “other” database table. At least one record was to have all fields populated. At least one record was to have null in all nullable fields (null values in the database can cause your code to throw exceptions if you have not handled things properly).
3. Navigation Requirements

Your home page shall have at least these three links in its navigation bar: Home, Angular (new), and Blog.

After adding in the New Angular link, the HTML code for your home page’s Nav bar might look like this:

```html
<div id="nav">
  <a onclick = "ajaxFillId ('htmlPartials/homeContent.html', 'content')">Home</a> &nbsp; | &nbsp; 
  <a href = "08_Angular/index.html">Angular</a> | &nbsp; 
  <a onclick = "ajaxFillId ('htmlPartials/blogContent.html', 'content')">Blog</a> &nbsp; &nbsp; 
</div>
```

When the user clicks on “Angular”, they shall find themselves in the new Angular Single Page Application (SPA) that is this week’s homework. Your Angular home page (SPA) shall have these three internal links in its nav bar: “NgHome”, “Users”, and a link you decide (that represents your “other” database table). The Angular home page shall have a 4th link (a “normal” link) “Back” that links the user out of the SPA and back to your main home page.

```
<div id="nav">
  <div><a href="#home">NgHome</a></div> <!-- Internal link stays in the SPA -->
  <div><a href="#users">Users</a></div> <!-- Internal link stays in the SPA -->
  <div><a href="#resorts">Resorts</a> <!-- Replace "#resorts" and "resorts" with the name of your other db table. -->
  <div><a href="../index.html">Back</a> <!-- Exits you from the Single Page Application, back up a level to the main home page -->
</div>
```

4. Styling Requirements

The layout (look and feel) of your Angular SPA (08_Angular/index.html) shall be similar to the layout from your main home page, but with some modifications so that you can tell which page you are on. For example, you might modify page background color and title of your Angular home page.

5. Project Organization Specs (For Routing)

Your project files shall be organized as shown – to get the Angular Routing part of the assignment to work.

Your project should be named something like “Smith_3344” and contain working functionality for this and all previous homeworks. Back up your project folder before starting this HW (back it up often – whenever you get to a stable version of code where things are working well).

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

Create a folder to hold “HTML partial” files. At first put in dummy content like “Users Coming Soon”.

Create a folder to hold javascript files.

Create copies of artifact folders like “pic” and “style” so that your Angular Single Page web site is self contained.

This is your Angular Home page (Single Page Application).

From here down your files should be as they were – implementing your main page’s functionality (home and blog).
6. Requirements for Click Sortable HTML Tables (with Client Side Filtering)

When the “Users” link is clicked, an HTML table shall appear with all of the data from your web_users table, similar to what is shown below. This HTML table shall provide Angular “click sortable” columns and client side search filtering functionality.

- If the user clicks any column heading, the rows in the HTML table shall be reordered to display in the order (or reverse order) of the clicked column.
- If the user types into the search box, the HTML table will automatically show only those records which include text that matches what is in the search box.
- It is best to not show the user passwords (and in a secure app, the passwords would not be included in the JSON provided by the getUsersAPI), but there will be no point deductions if you choose to display passwords on your page.
- Leave the error message column (even though a production application would not do so) – just to help us grade your work, so that we know there were no SQL errors in your java/JSP code.

Your Angular Home/index page shall have an “other” link which shall be named according to whatever you named your “other” database table). When this link is clicked, an HTML table shall appear with all of the data from your “other” database table. The functionality of this HTML table shall be like the functionality of the Web Users HTML table (Angular “click sortable” columns and client side search filtering functionality).

7. Additional Styling Requirements

You may have to modify the styling for your Angular Single Page Application. For example, I had to:

- Add a “white-space: nowrap” property/value to the rule that styled the HTML table column headings so that the sort icons would not wrap to the next line.
- Change the background colors of the <th> and <td> to blend better with the new page background color.
8. Overall Project Organization Requirements

The previous Project organization diagram showed you how to organize your client side artifacts to get basic routing working. This diagram includes server side artifacts (JSP pages that are Web APIs and the java files/packages that are referenced by the JSP pages). Your project artifacts shall be organized as shown below EXCEPT for these artifacts that shall be named differently:

- resortController.js and resortContent.html shall have names that are appropriate to the name of your “other” DB table.
- model.xxx and the second class in the view package shall both have names that are appropriate given the name of your “other” DB Table.

Your project shall contain the code from ALL previous homeworks (I am only showing 07_WebAPIs, but you need all HW folders). Your homework for this week (Angular SPA) shall be “self contained” inside of folder “08_Angular”.

You may want to create an “icons” folder to hold the sort icons that will be in the column headings of your HTML tables.

Organize your code by placing each JS controller in a separate JS file (well named). Your index file must reference each of the js files (or their code will not be available). In the sample code, routing_layout_app.js must be run first because it creates the Angular module object that is referenced by the other two js files (why I numbered them).

You may have to tweak the styling for your Single Page Angular App.

Copy both Web APIs from last week. Place them in a webAPI folder to keep things organized and manageable.

This index.html file is your Angular SPA (the only file you will run to test – other than testing/running your Web APIs directly).

Keep (or copy in) your source packages (containing java files) from last week. These are referenced by your Web APIs (JSP pages).

You’ll write the code in this package (StringData and StringDataList shall be similar to the same classes model.WebUser)

Don’t forget you need to add the two JAR files to the Libraries: JSON <-> POJO conversion and MySQL DB driver.
9. Blog Requirement

Do not forget to write a blog for your work this week, discussing what you found hard, easy, and important.

10. 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 that your work this week did not break any functionality from last week. **DO NOT have multiple projects locally that you publish/merge into one project on the web server.**
- Check that your blog page includes a blog for this homework.
- Confirm also that your blog for last week includes direct links to your Web APIs so that we can easily test this if we need to.
- After testing everything locally, 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 homework if you have trouble with publishing.
- Follow the "Requirements for All Homeworks and Projects" at the top of the 3344 web page.
11. Suggested Approach

a. **Study the Angular Tutorial** which explains and demonstrates about 8 different examples. Please note there is a zip file at the end of the tutorial which will enable you to run and experiment with the code.

b. **Do the Angular Intro Lab Activity** (if you have not already done that). In this lab activity you:
   - Created the 08_Angular folder with its own index.html and style sheet.
   - Implemented navigation between main home page (Web Pages/index.html) and Angular home page (WebPages/08_Angular/index.html).
   - Got basic routing working (with dummy content) in 08_Angular/index.html – for links “Home” and “Users”, remembering that:
     - You need two CDN <script> references (one to Angular and one to Angular Routing).
     - The ng-app directive (HTML attribute) must surround all the HTML that Angular needs to access AND the value of the ng-app attribute must match with the Angular code when you call angular.module() to create the Angular Application object.
     - The ng-view directive must be placed on the <div> where you want the content to be injected.
   - For the “Users” link, implemented a bullet list showing the web user data as provided by the “get users” Web API. To do this, you wrote a controller that made an AJAX call (using http-get) to the get users Web API and you updated the users HTML partial (using ng-repeat).
     - When creating multiple JS files, remember to name them well (similar to or exactly the name of the controller function they contain) and remember to reference the JS files from your index page (the JS file that creates the Angular module object must be run first since all the JS files will reference that one global object).
     - The JS code that creates the Angular Application object must be run first since all other JS code will access the global Angular Application object.

  c. **Back Up your Web App** – the project that has all your homework from the semester especially including the previous homework in which you wrote two Web APIs. To back it up, you can just copy your project folder from NetBeans Projects using File Explorer.

d. **Incorporate code from the Angular Intro Lab Activity** into your 3344 Web App (that has all the HW from all semester). 08_Angular should be “self contained” in that it has its own copy of style sheet, pics, Web APIs, etc. But there is only one copy of Java code that is under Source Packages and this will be shared by your Web API Homework and your two Angular Homeworks.

e. **Enhance Show User Functionality.** In the Angular Intro Lab activity, you used Angular ng-repeat to create an unordered list that showed the Web User Ids and email addresses (as extracted from your database by your “get users” Web API). For the HW, you have to change the web user <ul> to a <table> tag with <tr> tags (rows) and <td> tags (table data/cells). There is an example of a <table> created by ng-repeat in the Angular Tutorial Sample code.

  f. **Add the functionality for displaying the data from your “other” database table.** Create a new link in your NG home, add a new routing rule which references a new HTML partial and a new controller. The controller will invoke the “get other” Web API.

g. **Organize your project** as specified earlier in this document. If you rename or relocate a file, make sure to update the relevant part of your code.
12. Tips and Troubleshooting

a. Since you are new to Angular, be sure to always have working sample code (perhaps in a separate project). Identify the ways in which you want the sample code to be modified. **Run/test your code after each small change.** Once you get more familiar with Angular, you could become more aggressive and make more changes between testing. Whenever you get something working well, **back up** and name the backup so that you will know what it is.

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

c. **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...

d. If your **project is malformed** (you are getting weird compiler errors in NetBean’s output pane when you try to run a Web API / JSP page),

- Make sure that the Web API (JSP page that you are running directly or that is getting invoked by an HTML page) is well formed. I usually click on the first { and check that there is a matching } and no extra }. You can try right clicking your project (from the NetBeans project window) and selecting "Clean and Build" (to recompile everything). Make sure there are no red errors in the NB output window. You may have changed the name of a java class or package and not changed it in the Web API / JSP page.
- If that doesn’t work, you can create a new project (add the two JAR files first, then copy down the Source Packages from your old project, then the client side code (everything under “Web Pages” except for WEB-INF).
- If this does not help, maybe the installation of your NetBeans bundle is bad, but hopefully you have had plenty of time to test your NetBeans installation. To test the NB installation, just create a new project with a new JSP page and try to run that JSP page. If this works (launches browser and says Hello World), your installation is OK. If not, you may have to uninstall glassfish and NetBeans and any java versions that are newer than version 8, then reinstall java 8, then the NetBeans bundle.

e. 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.

f. **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.
g. 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.

h. 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.

i. 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 client side changes:
   - Hold down the Control key while you refresh your browser (it is supposed to not use cached items)
   - OR Right click and run index.html (instead of just refreshing whatever page you were testing).
   - OR 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).

j. 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) $\rightarrow$ JSON on the server side.
   - On the client side, JSON.stringify() and eval() / JSON.parse() converts javaScript objects $\leftarrow\rightarrow$ JSON.

k. 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).