AngularJS Intro Homework

Contents

1. Overview................................................................................................................................................2
2. Database Requirements .............................................................................................................................2
3. Navigation Requirements...........................................................................................................................3
4. Styling Requirements..................................................................................................................................4
5. Project Organization Specs (for the Routing Part of this Homework)....................................................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 for Routing.............................................................................................................7
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 a **Web API** (JSP page) that retrieved all the columns of all the records from your "Other" database table and displayed them in JSON format. Then you wrote some JavaScript code that invoked your Web API using a technique called AJAX.

*In this homework,* you will create a Single Page Application (**SPA**) contained in folder “08_Angular” in which you take advantage of some Angular’s features:

1. Routing (let’s you inject content into a layout instead of copying in “top.html” and “bottom.html” – your whole web site functionality will remain inside of index.html).
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, you’ll copy/paste listOtherAPI.jsp and listUsersAPI.jsp from last week’s folder into this week’s folder.

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 lab.

- Note: Your “other” database table is NOT to contain people. You have a web_user table was 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).
3. Navigation Requirements

Your main home/index and labs page shall work as it worked before (both using Angular to include “top.html” and “bottom.html” to achieve HTML code reuse). Your main home/index page shall have at least these three links in its nav bar: Home, Angular (new), and Labs.

Once the user clicks on “Angular”, they will find themselves in the new Angular Single Page Application (SPA) that is this week’s homework. Your Angular home page (SPA) shall have these links in its nav bar: “NgHome”, “Users”, [Other] (name like your “other” database table), and “Labs”. Of these links (see HTML below), only “Labs” will actually link to a whole new HTML page (up one folder, back to your main web root folder). All other links are internal page links (like we used in our Advanced Layout/Responsive Design Homework).

Staycations

Your web site will be built around the database design that you choose (check the database lab

<div id="nav">
    <div class='navLink'><a href="index.html">Home</a></div>
    <div class='navLink'><a href="08_Angular/index.html">Angular</a></div>
    <div class='navLink'><a href="labs.html">Labs</a></div>
</div>

Staycations Ng

This is my Angular single page "web site". Right now, it only lets you look at my web_user table and my

<div id="nav">
    <div><a href="#home">NgHome</a></div>
    <div><a href="#users">Users</a></div>
    <!-- Replace "#resorts" and "resorts" with the name of your other db table. -->
    <div><a href="#resorts">Resorts</a></div>
    <!-- gets you out of the single page application, back up a level to the labs page -->
    <div><a href="../labs.html">Labs</a></div>
</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 the Routing Part of this Homework)

Organize your client files as shown – to get the Angular Routing part of the assignment to work.

For help getting the routing part to work, see the section entitled “Suggested Approach”.

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.

Create a folder to hold javaScript files.

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

This is your Angular Home page – it DOES NOT include from top.html and bottom.html.

From here down your files should be as they were... Your main index page shall still use the Angular include functionality bringing in HTML code from top.html and bottom.html.

For help getting the routing part to work, see the section entitled “Suggested Approach”.
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.

![HTML Table Example](image)

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.
- Modify the selector for the <img> styling (in myStyle.css) to make it more specific – the <img> rule was making the sort icons huge. Instead of selecting all <img> tags (applying property width 100% to make the layout responsive), I changed the selector to only apply to images that were inside of the responsive columns:

```css
.width33Pct img, .width66Pct img { /* comma means applies to both things. space means “inside of”. So, this rule applies to all images inside of elements classed with width33Pct and all images inside of elements classed with width66Pc. */
  width: 100%; /* ... */
}
```
8. Overall Project Organization Requirements

Your project artifacts shall be organized as shown below. These are the 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 Angular Single Page Application for this week shall be “self contained” inside of folder “08_Angular” (under Web Pages in the NetBeans Project pane). This project shall contain ALL the code from ALL previous homeworks – I have not included all the Homework folders but you need them.

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

Place your JS controller code in separate JS files (well named) to keep your code organized and manageable. Your index file will need to reference each of the js files (or their code will not be defined/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 into their own folder to keep things organized and manageable.

This index.html file is your Angular SPA, the only HTML page you will run, other than testing your Web APIs (JSP pages).

Run your main index page to test navigation to the new Ng index page, and back to the main labs page.

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

Don’t forget you need the two JAR files: 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 labs page has blog for this lab.
- 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 lab if you have trouble with publishing.
- Follow the "Requirements for All Labs and Projects" at the top of the 3344 labs page.

11. Suggested Approach for Routing

a. **Rebuild Ng Index page** (copy/paste from top.html and bottom.html). After creating folder “08_Angular”, copy in index.html, you style and pictures folder as shown above. You’ll want to make your new index page be complete without referencing “top.html” and “bottom.html”, so reverse what you did in the first homework, copy from “top.html” and “bottom.html”, and paste into index.html where they were referenced. Make sure there is no “top.html” and no “bottom.html” in the 08_Angular folder. Run your index page (the new index page from 08_Angular) and make sure it renders OK with all the references to style etc. Remove the content from the content area - soon we will write js code that injects various content into the content area.

b. **Modify the Ng index page** slightly so that you can tell which page is which. You can change the background color and add and “Ng” to the title (“Ng” stands for aNgular).

c. **Get page navigation to work** from your main home page to the new Ng home page and from the new Ng page back to your labs page. We’ll get the internal links in the Ng index page to work later.

d. **Create the htmlPartial folder** and prepare the three files inside. You’ll need homeContent.html, userContent.html, and xxxContent.html (substitute “xxx” with the name of your “other” database table). Inside of homeContent.html, you can put the same content you had in your regular home page. Remember this is just an HTML snippet file – it does not start with <html> and end with </html>. It just contains the HTML that is to be “injected” into the content area of the new index page that’s in 08-angular. For your other two pages, just put some heading that says something like “user content coming soon”, “xxx content coming soon”.

e. **Run and study the example code** under the heading entitled “Using Angular Routing for Layout (UI Code Reuse)” from this page: [http://cis-linux2.temple.edu/~sallyk/tutorials_AngularJS/index.php](http://cis-linux2.temple.edu/~sallyk/tutorials_AngularJS/index.php).

f. To make your internal links work using Angular Routing, notice:

- Add the ng-app attribute to a DOM element that contains the whole user interface. Angular will not touch anything that is outside of the DOM element that has the ng-app attribute. Make sure your content area is inside of the DOM element that has the ng-app attribute.

  ```html
  <div ng-app="demoApp"> <!-- inside this div you can use Angular directives -->
  
  </div>
  ```

- At the top of your new Angular index page, reference angular as well as the angular routing library of minified javaScript code.

  ```html
  <head>
  <title>Staycations</title>

  <script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.2.10/angular.min.js"></script>
  <script src="//cdnjs.cloudflare.com/ajax/libs/angular.js/1.2.10/angular-route.min.js"></script>

  </head>

  ```

- Your new Angular index page must reference the javaScript code that invokes the Angular functionality:

  ```html
  <script src='js/routing_layout_app.js'></script>
  </body>
  ```
• The code in js/routing_layout_app.js must define all of the internal links that you defined in your ng index page.

<!-- Nav Links from the Ng Index page -->

<div id="nav">
  <div><a href="#home">NgHome</a></div>
  <div><a href="#users">Users</a></div>
  <div><a href="#resorts">Resorts</a></div>
  <div><a href="../labs.html">Labs</a></div>
</div>

// js/routing_layout_app.js

var demoApp = angular.module('demoApp', ['ngRoute']);
demoApp.config(function ($routeProvider) {
  $routeProvider.
    when('/', { // default link if index.html is not provided...
      templateUrl: 'htmlPartials/homeContent.html',
      controller: 'demoController'
    }).
    when('/home', {
      templateUrl: 'htmlPartials/homeContent.html',
      controller: 'demoController'
    }).
    when('/users', {
      templateUrl: 'htmlPartials/userContent.html',
      controller: 'demoController'
    }).
    when('/resorts', {
      templateUrl: 'htmlPartials/resortContent.html',
      controller: 'demoController'
    }).
    otherwise({ // if bad link, use default link (above) ...
      redirectTo: '/'
    });
});

// Each routing rule needs a controller, even if the controller doesn't do anything...
// We'll use these controllers in the second part of the homework assignment.
demoApp.controller('demoController', function () {
}); // end of def'n of the controller function
g. If your project is malformed (you are getting weird compiler errors), create a new project (add the two JAR files first, then copy down the Source Packages, 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. If so, you may have to uninstall glassfish and netbeans and any java versions that are newer than version 8, then reinstall java8, then the NetBeans bundle. To test, create a new web app project, add a JSP page to the web folder, run the newly added JSP page. If it launches a browser saying “hello world”, your installation is OK.

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

i. 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.

j. 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...

k. 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.

l. Only write a few lines of code between testing. Whenever you get something working well, back up and name the backup so that you will know what it is.

m. 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.

n. 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.
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).

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.

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).