CS142 Project 4: Page Generation with AngularJS

Due: Thursday, May 3, 2018 at 11:59 PM

Setup

You should already have installed Node.js and the npm package manager your system. If not, follow the installation instructions (install.html) now.

Create a directory `project4` and extract the contents of this zip file (downloads/project4.zip) into the directory. The zip file contains the starter files for this assignment.

This assignment requires a few node modules (e.g. JSHint (http://jshint.com/about) and ExpressJS (http://expressjs.com/)) that you can fetch by running the following command in the `project4` directory:

```
npm install
```

This will also fetch AngularJS (https://angularjs.org/) into the `node_modules` subdirectory even though we will be loading it into the browser rather than Node.js.

Like the previous assignments you will be able to run JSHint on all the project's JavaScript files by running the command:

```
npm run jshint
```

The code you submit should start with `"use strict";` and run JShint without warnings.

Your solutions for all of the problems below should be implemented in the `project4` directory.

This project uses AngularJS (https://angularjs.org), a popular framework for building web applications. The project's goal is to get you enough up to speed with AngularJS and the cs142's coding conventions that you will be able to build a web application with it in the next project.

Although AngularJS is written in JavaScript and runs in the browser we can not continue running from the local file system as we did in the previous projects. AngularJS handles much of the loading of the HTML for you and if you think about things from a security point of view the fact that your browser won't let random JavaScript code read files on the local file system is a good thing. The result is we will need to stand up a simple web server to serve your project files to your browser.

Fortunately, Node.js is a really good environment for writing simple web servers. We provided one that can be started with the command from the `project4` directory:

```
node webServer.js
```

All the files in the `project4` can be fetched using an URL starting with http://localhost:3000 (http://localhost:3000). Click on http://localhost:3000 (http://localhost:3000) to verify your web server is running. It should serve the file `index.html` to your browser.

Getting Started

In this project we require that you use the model, view, controller pattern described in class. There are many ways of organizing code under this pattern so we provide an example that both demonstrates some basic AngularJS features as well as showing the file system layout and module pattern we would like you to follow in your projects.

You should start by opening the example in your browser by navigating to the URL http://localhost:3000/getting-started.html (http://localhost:3000/getting-started.html). The page displays examples of Angular in action. You should look through the this web page since it shows with HTML statements we will be used to run an Angular web application along with explanatory comments.

Angular directly supports the model, view, and controller pattern we want. The line in `getting-started.html` containing

```
<html ng-app="cs142App" ng-controller="MainController">
```

effectively turns the contents of `getting-started.html` into an Angular template and starts the function `MainController` (found in `mainController.js`) as the controller for it.

To build reusable components we adopt a file organization that co-locates the template HTML, CSS, and controller of the component in a subdirectory of a directory named `components` . This can be seen in `getting-started.html` in the line:

```
<div ng-include="'components/example/exampleTemplate.html'" ng-controller="ExampleController"></di
```

that replaces the div elements with the HTML from `exampleTemplate.html` operating under the controller `ExampleController` . The lines in the `head` section:

```
<script src="components/example/exampleController.js"></script>
<link rel="stylesheet" type="text/css" href="components/example/example.css" />
```

read in the ExampleController and the stylesheet for the component. You should use this pattern and file naming convention for the other components you build for the class.

Model data is typically fetched from the web server which retrieves the data from a database. To avoid having to setup a database for this project we will give you an HTML script tag to load the model data directly into the browser's DOM from the local file system. The models will appear in the DOM under the property name `cs142models` . You will be able to access it under the name `window.cs142models` in an AngularJS controller.

Problem 1: Understand and update the example view (5 points)

You should look through and understand the `getting-started.html` and the `example` component. To demonstrate your understanding do the following:

1. Update the model data for the example component to use your name rather than "Unknown name". You should find where "Unknown name" is and replace it. **This is the only place where you are allowed to enter your name.**
2. Replace the `div` region with the class `header` in `getting-started.html` with your own design for a header that displays your name and a short (up to 20 characters) motto. You must include some

styling for this header in `main.css`.

3. Extend the example component so it allows the user to update the motto being displayed. The widget you use to perform the update should be placed in the `div` with the class `motto-update` in the example template file.
    - Note that the example component uses a different scope and controller than the main component. The example component's scope is a child of the main's scope and so uses JavaScript prototypical inheritance.
    - **The div with the class `header` should remain in `getting-started.html`.**
    - The `window.cs142models.exampleModel` data should remain in the example component. You should not have window.cs142models.exampleModel anywhere outside ExampleController.
    - Like the user's name, the initial value for `motto` should come in with the model data.

Problem 2: Create a new component - states view (10 points)

Create a new component view that will display the names of all states containing a given substring. Your view must implement an input field which accepts a substring. The view will display in alphabetical order a **list** of all states whose names contain the given substring (ignoring differences in case). For example, the view for the substring of "al" should list the states Alabama, Alaska, and California. The page should also display the substring that was used to filter the states. If there are no matching states then the web page should display a message indicating that fact (rather than just showing nothing). All states should be displayed when the substring is empty.

As in Problem #1 we provide you the model data with states. It can be accessed via `window.cs142models.states` after it is included with:

```
<script src="modelData/states.js"></script>
```

See `states.js` for a description of the format of the states data.

To help you get started and guide you to the file naming conventions we want you to use we provided a file `p2.html` that will load and display your component like `getting-started.html` does for the example component. You can open this file in your browser via the URL http://localhost:3000/p2.html (http://localhost:3000/p2.html).

The files you will need to implement are:

- `components/states/statesTemplate.html` - The Angular HTML template of your states component.
- `components/states/statesController.js` - The controller of your component. It should be added to the `cs142App` under the name `StatesController`.
- `components/states/states.css` - Any CSS styles your component needs. **You must include some styling for your state list here.**

We give a start with these files that add the Angular controller to the application.

Problem 3: Personalizing the Layout (5 points)

Create a component that will display a personalized header at the top of a page. Add this header to all pages in your assignment (`getting-started.html`, `p2.html`, `p4.html`, `p5.html`). This header is in addition to the header from part 1 - do not replace that header. Use your imagination and creativity to create a header that is "uniquely you". This can include additional images, graphics, whatever you like. You can extend the JavaScript in the controller but you may not use external JavaScript libraries such as JQuery. Be creative!

The files you will need to implement are:

- `components/header/headerTemplate.html` - The Angular HTML template of your header component.
- `components/header/headerController.js` - The controller of your component. It should be added to the `cs142App` under the name `HeaderController`.
- `components/header/header.css` - Any CSS styles your component needs. **You must include some styling for your header here.**

Note: `getting-started.html` should have 2 headers: the personalized header from Problem 3 at he top of the page and the header with the motto from Problem 1.2 right below it. All other pages(`p2.html`, `p4.html` and `p5.html`) should only have your personalized header from Problem 3.

Problem 4: Add dynamic switching of the views (10 points)

Create a `p4.html` that includes both view components (the `example` and `states` view components). The `p4.html` template and its `MainController` needs to implement an ability to switch between the display of the two components. When a view is displayed there should be a button above it that switches to display the other view. For example, when the `states` view is displayed the button above it should read "Switch to example," and when pushed the `states` should disappear and the `example` view should be displayed.

Problem 5: Single page app (5 points)

Although the approach taken in Problem 4 allows you to switch between the two views, it does not allow you to bookmark or share a URL pointing at a particular view. Even doing a browser refresh event causes the app to lose track of which view was being displayed.

We can address this deficiency by storing the view information into the URL. Angular provides a module that makes this easy to do. For this Problem make a copy of your `p4.html` solution into a file named `p5.html` and convert it to use Angular's ngRoute (https://docs.angularjs.org/api/ngRoute) module to switch between the two component views. You should have a **styled toolbar-like control** (simple plain text links are not sufficient) that will allow the user to switch bewteen the example and states component views.

Since this is the first Angular module we import we provide you with step-by-step instructions.

1. Angular keeps modules in separate files so you will need to explicitly load the ngRoute module using a script element in the head section of `p5.html` like so:

   ```
   <script src="node_modules/angular-route/angular-route.min.js"></script>
   ```

2. We need to inform our angular application that it needs ngRoute by adding it to the currently empty dependency list by changing the angular.module called in `mainController.js` to:

   ```
   var cs142App = angular.module('cs142App', ['ngRoute']);
   ```

   Note that once you modify the dependencies, any HTML page using `mainController.js` must also have the ngRoute module loaded into the head section.
3. Rather than uses ng-if conditions to control the different views, you can simply add a `div` element with an attribute of ng-view. The ngRoute module will insert the component view into this region.

   ```
   <div ng-view></div>
   ```

4. We now need to configure the `$routeProvider` to know about the mappings from URLs to view comments. The following code added to the `mainController.js` will do this.

```
cs142App.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/example', {
        templateUrl: 'components/example/exampleTemplate.html',
        controller: 'ExampleController'
      }).
      when('/states', {
        templateUrl: 'components/states/statesTemplate.html',
        controller: 'StatesController'
      }).
      otherwise({
        redirectTo: '/example'
      });
  }]);
```

5. To access the views you need to specify the fragment on the URL. For example:

```
<a href="#!example">Example</a>
<a href="#!states">States</a>
```

6. **Make sure you check that Problems #1-4 still work with your updated** `mainController.js`. In particular, you'll have to load the ngRoute module into the other 3 HTML files using a script like you did for `p5.html`.

Style (5 points)

These points will be awarded if your solutions have proper MVC decomposition and follow the Angular style guidelines discussed in lecture and section. **Note that you should not directly manipulate the DOM in your code. Use Angular's directives to achieve the functionality you want.** In addition, your code and templates must be clean and readable. Remember to run JSHint before submitting. JSHint should raise no errors.

Deliverables

After implementing part 5, make sure parts 1 through 4 still work!

Use the standard class submission mechanism (submit.html) to submit the entire application (everything in the `project4` directory).