CS275 Web and Mobile App Development
Winter 2017
Lab 4

As with this and all future labs, you must work individually.

Objective:

The purpose of lab four is three-fold:

1. Allow for multiple pages with a mixture of static and dynamically generated content.
2. Begin applying good software design principles by segmenting logic into their own modules.
3. Get experience playing with Nodejs emitters.

Note: We're going to be porting lab 2 and lab 3 into this lab. So if you were unable to get lab 3 and/or lab 2 (which was related to the aforementioned example) working properly please see an instructor or TA ASAP to get these working according to specifications.

Part 1: Create your "home page"

In Nodejs, if we set up our express server to serve up static files, then requests to the root of the server (http://localhost:<port>) will attempt to read (and respond with) a file called *index.html* if no handler for the endpoint / is defined.

So let's start this lab by creating a file called `index.html` that has the following components:

- Some sort of "banner" at the top
- A navigation area on the left (it's column will constitute 20% of the page)
- A "content" area to the right of the navigation area (and therefore constitutes 80%) of the page.

The banner can be whatever you like. Perhaps an image or some text.

The navigation area should have three links entitled something like:

1. Home
2. Calculator
3. Weather

The content area should initially give instructions on how to use the site and what it does (you'll probably populate this after you get the remaining parts done).

Here's a crude idea (feel free to make it nice looking)

Part 2: The links

Your links should be as follows

- The home link just sends the user to the root homepage, i.e './'
- The calculator link, when clicked, will populate the content area (asynchronously) with stuff related to lab 3 (more on this in part 3)
- The weather link, when clicked, will populate the content area (asynchronously) with stuff related to lab 2 (more on this in part 4)

Notes/Hints: If your `href` path is just a pound sign, #, then clicking on the link won't redirect you to another page (it will just take you to the top of the current page).

Part 3: The Calculator

Ok. On to the tough/fun stuff! We're now going to port your lab 3 (the one that did computations) into this lab, but we're going to have its rendering and computations done via its own module! The idea/flow is as follows:

1. When the user clicks on the Computer link, it loads the html content (scripts and body) you generated in lab 3 into the content area (asynchronously).
2. Then you should be able to use the content area just like you did in lab 3.

But we're going to organize this to have these things done via a `Calculator` module:

- Create a `Calculator` module where you export several functions, in particular:
  - `render`
  - `computeFactorial`
  - `computeSummation`
- Your main server script should then require this module and direct endpoint requests to the appropriate methods. For example:
  - When the Computer navigation link is clicked, your corresponding endpoint should call your module's render function, populate the HTML string containing the scripts/body from Lab 3, and return this to the endpoint so that it can send it back in its response.
  - When the button is clicked in this rendered view, your code (like in Lab 3) should call some endpoints on the server, which in turn call the corresponding functions of your module, which does the appropriate computation and returns the HTML data to be used by the AJAX call.

Notes/Hints:

- To have multi-line JavaScript strings (which you may want, especially for your rendering method) the best way is to just have each line concatenate to the total string. For example:

```
var str = "<html>" +
"<head>"+
//etc..
```

- Also remember that you might have to escape quotes within your string.
- You could either do this by creating a class and binding functions to it, or by just exporting various static functions, the latter of which we described above.

Part 4: Get Weather

In Part 3 you got to play with modules and exporting functions. Now we want to play with emitters.

Let's port your lab 2 so that the server does a lot of the leg work! Here's the details:

1. Store your weather underground key in a file on your server, one directory up from where the server is running (for security)
2. Create your `Weather` module that has the following functions/methods
    a. `Render`
    b. `getZip`
    c. `getWeather`
3. Within this module, read in and store your key from your file.
4. When the user clicks on the Weather link in the navigation area, you should render the body (including the necessary scripts) of your HTML code using the text returned by the `render` method. This should basically just show a button (since you have the key on your server).
5. When the user clicks the button, an asynchronous request is sent to the server which:
    a. Binds a listener to your Weather module for something to be emitted by your `getZip` method
    b. Calls the `getZip` method of your Weather module
        i. This method makes an HTTP request call to weather underground
        ii. Once it has all of its data, it extracts the zip code from the response and emits a signal saying it's done.
    c. Once your listener gets the emitter signal from the `getZip` method
        i. Grab the zip code data that was passed to your bound function.
        ii. Binds a listener for something to be emitted by your `getWeather` module.
        iii. Call your `getWeather` module
    d. Your `getWeather` method should
        i. Construct the necessary URL using the zip code and your key and makes an HTTP request call to weather underground.
        ii. Upon completion, construct an HTML response from the data it obtained and emit a signal indicating you're done, along with this HTML data.
    e. Once your listener gets the emitted signal from the `getWeather` method, send your response with the returned HTML data in it.

Notes/Hints: You can get a lot of this from the Node Part 3 slides

What to Submit

- A screen cast video to detailing a thorough (both client and server side) code walk through, followed by a demo of all cases
- Your source code, well internally documented
- README file on how to run your code
- All of the above **ZIPPED** together and that single compressed file uploaded to BBlearn.

Grading (50) Points

- 40 points: program correctness along with adherence to the stated requirements
- 5 points: quality of internal documentation and code style
- 5 points: README file