SER421 Lab 3 – Node and Express   Due: 11:59pm, Thursday, October 11

Activity 1: Put your calculator online! (30%)

In lab 1 you constructed a calculator that processed incoming JSON expressions. For this activity you will construct a simple Express application putting your calculator online. The requirements are a mix-and-match of Parts A and C of Lab 1, Activity 2:

R1. The main UI of your application (/) is a page that displays: *(6 - 2 pts each a-c)*
  a. The current value of a calculator (0 when it first starts)
  b. A stack view *of the history of the operations* applied to the calculator (see R7).
  c. A simple web form that accepts input of a number with options to "Add" "Subtract" or "Pop"

R2. As inferred by 1, your calculator should support 3 options: Add, Subtract, and Pop *(9 - 3 pts each for a-c)*
  a. Add must be implemented under /add and only support POST. It adds the operand to the present value of the calculator and stores the operation in the stack.
  b. Subtract must be implemented under /subtract and only support POST. It adds the operand to the present value of the calculator and stores the operation in the stack.
  c. Pop must be implemented under /pop and support GET or POST. It pops the top operation off the stack and changes to present value of the calculator to the previous value. NOTE: Read the requirement 1c and 2c carefully.
  d. The response to each of these actions should be a message stating success (or an error, see C2) and a link back to the landing page (/).

R3. Implement a 4th option "Reset" that supports only GET at URL endpoint /reset. This operation resets the calculator to a value of 0 and empties the stack. Again, the response should be success or error. *(2 pts)*

R4. Push is inferred; you are to automatically Push all successful Add or Subtract operations; however you do not have a URL to expose Push as an operation invoked from the UI. *(2 pts)*

R5. You should ensure that pages are not cached. Each request to the server should be "new". *(3 pts)*

R6. You must validate the URLs, methods and inputs your program receives. If an invalid URL is given, ensure the proper HTTP error code is returned. If an invalid method is given to the URL, ensure the proper HTTP error code is returned. If the input typed into the form is invalid, ensure the proper HTTP error code is returned. *(5 pts)*

R7. The stack implementation is a history or operations applied to the calculator, not a memory of results as in lab 1. How you represent the history of operations is up to you, but it must include the operation, the operand (number), the IP address and the User-agent of each request. *(3 pts)*

Constraints:

C1. You must use one of Pug or EJS to render each page. *(-5 if you hardcode HTML in your JS)*

C2. No Javascript or CSS in the browser at all. *(automatic -5 and 0 for any requirement above affected)*

Notes:

1. You may use your solution to lab 1 or use ours. Note that lab 1 used JSON expressions, and here you are merely using form input. Also note there are no nested expressions (the calculator is like Part A). You may consider reusing Part C but you will have to modify the stack (again, read 1c and 2c carefully) and take out parsing of nested expression.
2. You will be graded in part of properly identifying Express routes to use and proper processing of route parameters.
3. The calculator is shared; that is there is no requirement to compute different values based on different users/browsers. All users of the app share the same calculator state.

Activity 2: e421Match.com: Find your next SER421 class partner online! (70 points)

You are asked to develop a web application that determines who should be your next SER421 class partner based on a series of questions. The app works like this: A user comes and enters her/his name on the site's landing page (/). S/he then takes a survey about their 421 preferences. At the conclusion of the survey s/he receives a ranked list of users most compatible by comparing their answers to the answers of all others' surveys. Users may return later and change their answers and receive a new ranked list of partners.

An example of this application is available at http://swent1linux.asu.edu:8082/simplemvcex. Your task is to replicate this program on the Node and Express platform, with these updated requirements:

R1. Implement a one-shot timer that terminates the survey immediately if the user has not completed it in 60 seconds.

R2. The last page currently redirects back to the homepage, you can merely provide a link back to it.

R3. Once you match your route and method in Express, put the code you delegate to (think Step 3 of the template pattern we have talked about) into files separate from the main routine. The main routine file should be app.js, and you should provide a package.json specifying external dependencies with targets to install and run your application

R4. Be sure to implement error handling appropriate to your application (as in Activity 1 R6 above).

R5. You should ensure that pages are not cached. Each request to the server should be "new".

Constraints:

C1. Use Express routes to map the "handlers" delegation logic to URLs. You decide the endpoints and appropriate methods to respond to. Document your design in your readme.txt.

C2. Use Pug or EJS templates to render each page of the application. *(-5 if you hardcode HTML in your JS)*

C3. You must use session middleware to accomplish the conversational state features of this app. Do not use cookies, hidden form fields, or URL rewriting for *conversational* state.

C4. Use the 'fs' module to read survey questions and write individual user responses. Past user responses (if available) should be pre-populated on the web forms, even if the user logs in via a second browser. You design the file formats used to manage survey questions and individual responses. NOTE: The number of questions in the survey should be variable – do NOT hardcode the set of questions based on this example! They should go in a file with a well-defined format (text, xml, json, whatever), and if the set of questions in the file changes, then the questions in the survey should change.

C5. No Javascript or CSS in the browser. *(automatic -5 and 0 for any requirement above affected)*

Notes:

1. You do not have to match the URLs in the example application (it was written using servlets). You may come up with your own URL scheme as part of your design and document it in your readme.txt.

Rubrics:

1. R1: 5 points
2. R2: 2 points
3. R3: 10 points
4. R4: 5 points
5. R5: 3 points
6. Handling an arbitrary number of survey questions from the file. (5 pts)
7. Proper conversational state management of the survey using Express middleware
   a. Carrying survey questions forward through the survey (6 pts)
   b. Properly handling the "Prev" button (6 pts)
   c. Pre-populating survey responses from any browser when re-logging in (8 pts)
8. Match functionality: 10 points
9. Horizontal/Vertical rendering options: 10 points

Extra Credit:

EC1. (10) Use async/await to perform I/O asynchronously in Activity 2.

EC2. (15) Use a database as the persistent store for survey answers instead of a flat file. You may use MongoDB or you may use MySQL (with the NodeJS to MySQL driver here: https://dev.mysql.com/downloads/connector/nodejs/). Do not use a ORM framework such as sequelize. If you decide to do this EC then you do not have to implement file persistence for the survey answers part of Activity 2 if you do not wish to (but then you have to get this EC correct to get full credit for Activity 2!). Note that the survey questions must remain file-based.

Submission instructions:

1. Submit your solution in a zipfile named asurite1_lab3.zip. In that file should be 2 source trees in 2 subdirectories, /activity1 and /activity2.
2. Each source tree may have its own readme – in fact you must have one for Activity 2 to inform us of your design.
3. Please listen on port 8088 in your http servers.
4. Do not add extra modules beyond those we have discussed in class or the videos (e.g. url, querystring, fs, express, middleware) unless you get permission first.
5. No Javascript and CSS in the browser at all in this lab.
6. If you decide to do EC2, be sure to provide any database initialization scripts we need to run with a description in the readme of eactly what preparation we need to do.

See attached.

For EC: NodeJS MySQL driver page:

https://dev.mysql.com/downloads/connector/nodejs/