Node.js and Express

LAB GUIDE

HTTP Module

1. Create a folder, say my-website. Initialize it as a Node.js project. Create 2 pages
* index.html that includes a few lines about yourself
* contact.html that has your contact details like email id, phone number, and a contact form that takes in user's name, emailid, and message.

Have these pages served on port 3000 using an HTTP server.
* GET http://localhost:3000/ returns the index.html page
* GET http://localhost:3000/contact returns the contact.html page
* POST http://localhost:3000/message accepts the details from the contact form on submission and writes them to a messages.json file (design a schema to store the data)
* Each page should have a menu on top to switch between index page and contact page.

Hints:
* req.url has the URL from incoming request. Decide which file to serve based on it (req is the request object here).
* req is a readable stream (so you can listen for data event on it). Concatenate the chunks received to form the request body. Then parse it to get message details.
* Check the documentation to find out how to retrieve the incoming request's method (GET/POST etc.)

2. Create a simple HTTP server that runs on port 3000, does simple arithmetic operations, and responds with the result.

Some examples
* http://localhost:3000/add?x=12&y=13 returns the string 25
* http://localhost:3000/multiply?x=12&y=13 returns the string 156

Your server must support add, subtract, multiply and divide operations. For an unsupported operation/arguments it must return a sensible error message, with appropriate error code (HTTP status code 401 is for a badly constructed request)

3. Create a simple HTTP server that takes a URL as input, and generates a shorter URL (just like a [URL shortening](https://en.wikipedia.org/wiki/URL_shortening) service like bit.ly does).

Example
POST http://localhost:3000/shorten with request body https://en.wikipedia.org/wiki/Design_rule_for_Camera_File_system should return a unique short URL, says http://localhost:3000/z34DCs. The algorithm and format of exact URL generated is upto you, but generated URL must be unique, and your algorithm MUST always generate the same short URL for a particular URL (the "long" URL). i.e. if the same request is made again, the short URL returned must also be exactly the same.

Finally, when you make a GET request for the short URL, the server must send a redirect request - this is achieved with status code 304, and adding a Location header whose value is the complete URL (the "long" URL). For more information on redirects check [this page](https://en.wikipedia.org/wiki/URL_redirection).

FS Module and HTTP Module

4. Create a json-utilities module (file) that supports the following APIs

* Search for a particular key in a JSON file with a set of properties at the top-level, and get its value.
* Set the key to a particular value in a file similar to the one just above.
* Search for an item with a particular key-value pair in a JSON with an array of objects (items).
* Add/remove/edit items in a file similar to the one just above.

Use this module's APIs in another module. Also use this to make the solution to question 3 simpler.

5. Create a simple file server. The server by default lists all the files/folders in the folder within which it is run. For example, if it is run from the temp folder, then navigating to http://localhost:3000/ should list all the files in the temp folder.

Each listed file/folder is a hyperlink. When a file/folder is clicked, the URL should change to append the relative path to the file/folder. For example, if http://localhost:3000/ list xyz.html, then clicking xyz.html entry results in the request http://localhost:3000/xyz.html (i.e. URL in browser changes to it) – this obviously returns the file contents as response. In case a folder, say docs/ is clicked, the URL changes to http://localhost:3000/docs and the (clickable) list of files and folders in docs/ folder is returned as response.

Extra credits:

* You can also have a .. entry in the folder listing. Clicking it takes the user to the parent folder listing.
* Give an option to the user to create a new file/folder, under the current folder, through the UI.

Modules

6. Create a module to calculate areas of various shapes.
   - Square (public function) - makes use of Rectangle function to calculate area
   - Rectangle (private function)
   - Circle (public function)
   - PI (private variable)

   Make use of this module in another module (i.e. file)

7. Explore the chalk module on npmjs.com - install it and use it.

Simple API server with JSON File-based store

You will build a simple API server that serves a list of workshops, and allows you to add a workshop.

Use the provided workshops.json file for this exercise. This has a JSON array of sample workshops. You can use this as the initial set of data when the server starts up. Every workshop has a unique id and more details.

Create a simple HTTP server that serves the following

1. GET /workshops
   This returns a JSON array of all workshops.

   A note on how to do this
   Read from workshops.json and serve it.

2. POST /workshops
   This adds a new workshop object using the JSON data sent in HTTP request

   A note on how to do this
   Use the request object's (of type IncomingMessage) 'data' and 'end' to read the request body. Use JSON.parse() to convert it to a JS object (check reference provided below). Check if all expected properties of a workshop have been passed in the body, else respond with an appropriate error message, status code and Content-Type header set to 'application/json' (check reference on ServerResponse object's writeHead() provided below). Generate a unique id for every product (use any custom logic you deem fit, but id has to be unique), and adds it to the workshop object as the id property. Update the workshops.json file on the disk (overwrite with updated array of workshops). It is recommended you also maintain an in-memory copy of the workshops.json file as well for responding faster to user request (although in cases like power failure during a write to file this can lead to data loss issue).

   Reference for using 'data' and 'end' events to read request body
   https://www.tutorialspoint.com/parsing-request-body-in-node

   Reference for setting response status code, status message and HTTP headers
   https://nodejs.org/dist/latest-v14.x/docs/api/http.html#http_response_writehead_statuscode_statusmessage_headers

   Create some helper functions to solve small pieces of this exercise. Make sure to set up unit testing using Jest (or any other library of your choice), and test these helper functions.

Express JS

1. Create a middleware that logs the request method, user-agent, and time at which request is received, and response is sent, and total time taken to process the request.
   Bonus: Log the IP address from which request was received as well.
   Note: User Agent indicates the type of client (eg. browser) that makes the request. It is a standard HTTP header sent by browsers when making a request.

2. Recreate question 1 from Node.js exercises (my-website) using Express

3. Recreate question 2 from Node.js exercises (arithmetic operations) using Express.
   NOTE: There should be a single handler for the requests - an action path param (/:action) maintains the action.

4. Recreate question 3 from Node.js exercises (URL shortener) using Express. NOTE: On receiving request for a short URL, you will need to use Express' res.redirect() to redirect users to the long URL.

5. Recreate question 5 from Node.js exercises using Express and the Express static file server (do not build your own logic to serve the static files).

6. Recreate the simple API server with JSON file-based store using Express

7. Use Morgan - https://www.npmjs.com/package/morgan to log all incoming request details to a file (say server.log) in Apache combined format. Additionally, requests that result in an error response must additionally be logged in a separate file (say errors.log). You can also explore Winston - https://www.npmjs.com/package/winston which is a highly popular alternative.

8. Often user session information is maintained using cookies (session id which is generated when user logs in, and then set as cookie, and user details maintained as a map from session id to the user details on server-side).

   You can understand about cookies here
   - https://www.youtube.com/watch?v=rdVPflECed8
   You can understand the differences between various ways of maintaining data on the client side, i.e. browser (cookies, local and session storage) here
   - https://www.youtube.com/watch?v=GihQAC1I39Q

   Note: There are other options for data storage on the browser too - Web SQL (deprecated) and Indexed DB - but these are usually not necessary, unless the use-case is quite advanced (eg. complex offline applications).

9. Now that you understand cookies, you can use them in a Node/Express App. The following videos explain how you can use cookies, and also build an authentication system using cookies (instead of using JWT token).
   https://www.youtube.com/watch?v=2so3hh8n-3w
   https://www.youtube.com/watch?v=wEbs7KzaESg

   Aside: If you want to see how cookies may be set without using Express cooke parser middleware, this video starts off taking that approach
   https://www.youtube.com/watch?v=RNyNttTFQoc