Node JS and Express

LAB GUIDE

For each of the exercises/assignments/projects, create a new folder and initialize it as a Node project. Install all the dependencies locally.

NODE JS

EXERCISES

NOTE: These questions would typically take 10 – 20 minutes to solve.

MODULES

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

2. Create a module shapes.js 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)

3. Use the chalk module - https://www.npmjs.com/package/chalk to create a module called logger.js. It defines utility functions for logging messages with different colors. It should define the following methods and export all of them (i.e. all are public)
   - success(message) - logs the received message to the console in green
   - warn(message) - logs the received message to the console in yellow
   - error(message) - logs the received message to the console in red

   Import (using named export) the logger module in another module and demonstrate the use of each of the methods. For example, the following should log the message in green.

   ```
   success( 'Added a product to the database' );
   ```

   Also try to import the whole exported object and demonstrate its use (Assuming Logger represents the exported object)

   ```
   Logger.error( 'Unable to connect to the database' );
   ```

OS MODULE

4. Using the built-in os module of Node, do the following
   a. List the number of CPU cores and their details
   b. The total memory (RAM) and free memory. Calculate them in MB and print (1 MB = 2^20 bytes).

Note: The free memory that Node JS gets to see is different from the actual free memory. Refer https://stackoverflow.com/questions/51911849/node-js-os-freemem-do-not-work-correctly

c. The username and home directory of the logged-in user

PROCESS MODULE

5. Using the process object of Node, do the following
   a. List the environment variables
   b. List the value of the PATH environment variable. What does this variable represent in the OS/shell?
   c. Set a PORT environment variable in your system (to a value like 3000) before you run your script. List the PORT variable.
   d. Use process.stdout.write() to log to the console (instead of using console.log)

PROCESS MODULE – COMMAND LINE ARGUMENTS

6. Write a script that accepts three command line arguments (apart from ```node``` and the name of the script itself). The first of these 3 is an operation - one of ```add```, ```subtract```, ```multiply```, ```divide```. The last 2 are numbers. The scripts runs and prints the result of the specified operation performed on the 2 numbers. For example, running the following, prints what is shown below.

   ```
   node operation.js multiply 4.5 5
   22.5
   ```

   Perform appropriate error handling (number of arguments is incorrect, specified operation is invalid, or the last 2 arguments are not numbers). In this case display a helpful message describing the correct way to provide input.

   Note: Even though we may view the last 2 arguments as numbers, when read from process.argv, they will be string. We need to convert the string to number before using it.

PATH MODULE

7. Write a script that lists the name of the file (the script itself), and folder name within it exists. Use path module's join() method to display the complete path of the parent folder.

   Hint: The string '..' can be joined with the current folder path to get the parent folder path.

FILESYSTEM (FS) MODULE

8. Write a script that prints the contents of itself (reads the file contents of itself and prints it).

9. Write a script that reads a file's contents and writes it to a new file (i.e. it copies a file). Use readFile() and writeFile().

   Note: You can create a sample hello.txt with some content, and copy it into another file called hello.copy.txt using this script.

10. Write a script that reads a file's contents and writes it to a new file (i.e. it copies a file). Use readFileSync() and writeFileSync(). Which approach is preferred in general? Async (eg. using readFile/writeFile) or sync (eg. using readFileSync/writeFileSync)? Why?

Note: You can create a sample hello.txt with some content, and copy it into another file called hello.copy.txt using this script.

11. Write a script that reads a file's contents and writes it to a new file (i.e. it copies a file). Use createReadStream() to create a read stream and read the file. Use createWriteStream() to write to the copy file. Which approach is preferred in general? Async without streams (eg. using readFile/writeFile) or using streams (eg. using readFileStream/writeFileStream)? Why?

    Note: You can create a sample hello.txt with some content, and copy it into another file called hello.copy.txt using this script.

HTTP MODULE

12. Create a simple HTTP server that runs on port 3000, does simple arithmetic operations (add, subtract, multiply, divide), 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

    For an unsupported operation/arguments it returns a sensible error message, with appropriate error code (HTTP status code **400** is for a bad request)

ASSIGNMENTS

NOTE: These questions typically take more than 30 minutes to solve.

HTTP Module

13. 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**
        - has your contact details like email id, phone number
        - has a contact form that takes in user's name, emailid, and message. \

    Each page should have a menu on top to switch between pages (index page and contact page).

    Have these pages served on port 3000 using an HTTP server.
    - GET **http://localhost:3000/** serves the index.html page
    - GET **http://localhost:3000/contact** serves 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)

    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.)

14. Create a simple HTTP server that takes a URL as input, and generates a shorter URL (just like a URL shortening service - https://en.wikipedia.org/wiki/URL_shortening. Example of such a service - https://bit.ly

    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 - https://en.wikipedia.org/wiki/URL_redirection

FS Module and HTTP Module

15. 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 14 simpler.

16. 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.

MINI PROJECTS

NOTE: These questions typically take more than 2 hours to solve.

17. Simple API server with JSON File-based store
    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
    - GET /workshops
      This returns a JSON array of all workshops.
      Note: Read from workshops.json and serve it.

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

    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.

    Hints
    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

EXPRESS JS

EXERCISES AND ASSIGNMENTS

NOTE: These questions would typically take 10 – 60 minutes to solve.

18. 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.

19. Solve question 13 (my-website) using Express

20. Solve question 12 (arithmetic operations) using Express.
    Note: There should be a single handler for the requests - an action path param (/:action) maintains the action.

21. Solve question 14 (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.

22. Solve question 16 using Express and the Express static file server (do not build your own logic to serve the static files).

23. Solve question 17 (simple API server with JSON file-based store) using Express

24. 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).

25. Solve question 24 using Winston - https://www.npmjs.com/package/winston which is a highly popular alternative to Morgan.

26. 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).

27. 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 cookie parser middleware, this video starts off taking that approach
    https://www.youtube.com/watch?v=RNyNttTFQoc