|  |  |
| --- | --- |
| Sprint 15, Assignment 3.8 Please also update the doc name with correct numbers. | |
| Assignment type: JS Interactive | |
| Assignment name  Can remain the same as the assignment topic name, or…  can be phrased in the active/imperative voice(depending on the task), e.g. “Create your own X” | “Create a user database script with Node.js” |
| Description  The essential tasks a student must perform in order to complete the assignment | A client has asked that you develop a simple interface for storing users in a JSON database. You've already learned how to access node's filesystem, as well as write code that relies on multiple files in that filesystem - so it's time to write a script that will allow the client to do this database work using simple console commands! You should deliver a script that, when run with the 'node', can be accompanied by an 'add' or 'delete' operation + a piece of incoming data (a new user name, or a current ID number etc) and will update the JSON database file accordingly. |
| Link to full code file (Answer)  Push a full code file to our [GitHub repo](https://github.com/bitdegree-foundation/academy-assignments-code-files) for this sprint’s assignment and paste a link to that file here. Please write the file in a concise and clear format, according to the “[example-js-code-file](https://github.com/bitdegree-foundation/academy-assignments-code-files/blob/master/example-js-code-file.js)” found in our repo. |  |

|  |  |  |
| --- | --- | --- |
| # | Step  Write each small step of the task. These assignments should be doable by students on their own (without any BitDegree instructor help). It can be any number of steps, but keep them pretty clear and separate (don’t combine 2 steps into 1 step). | 1x hint  Write brief text or partial code that would help the student figure out how to move forward or accomplish the given step. |
| 1 | In your index.js file (inside your "node-file-system-test" folder), 1) turn off your multiplication log command, and 2) import your node filesystem asynchronously | Hint: use "fs/promises" |
| 2 | Create a new json file named "users". In that file, add at least one ID number and one name (in the form of an object inside of an array) | Hint: [{"id":1}] etc |
| 3 | (In your index.js file) write an asynchronous function called "loadData" that 1) declares a constant called "data", 2) [a]waits for the users.json's data to be read, and 3) stores that data in the constant | Hint: use "await"  Hint: use node fs's ".readFile" method |
| 4 | Check what you are storing: 1) log the "data" constant \*after\* parsing it from JSON, 2) invoke your loadData function at the end of the file, and 3) then use node to run the file (you should see your user database entry | Hint: use JSON.parse  Hint: invoke the function with "loadData();" |
| 5 | Make the loadData function return the data (as well as log it) |  |
| 6 | Declare a new, second asynchronous function named "run". In the function, declare a "let" variable named "data" and have it store what's returned from loadData. Make sure it [a]waits for loadData to finish before storing. Also, change the invocation at the end of your file to run your new "run" function | Hint: use "await"  Hint: change the invocation from "loadData()" to "run()" |
| 7 | (Within your 'run' function) declare two constants named "process" and "incomingData", and store the appropriate argument values in them. Log the contents of these two variables, and run "index.js add myName" to make sure they're storing the correct values. | Hint: running your file will always, at the very least, supply the first two values - so you will need to store and use the ones supplied after  Hint: recall lesson 3.1's video and 3.2's text (https://nodejs.org/en/knowledge/command-line/how-to-parse-command-line-arguments/ ) |
| 8 | Add a conditional to your 'run' function that, when an 'add' operation is supplied by the user, will run an "addUser" function (to be written in the next step of this exercise). Make sure that you pass both the current user 'data' and the new incomingData to this addUser function | Hint: your conditional should evaluate whether your variable 'operation' is storing a string-version of "add"  Hint: the current user data is being imported via the 'loadData' function, and stored in a variable named 'data' |
| 9 | Declare a new function (after your 'run' function) called "addUser" that will 1. take in our current user.json 'data' and the newUser data, 2. declare's a constant named "newId", and 3. pushes that newId & newUser name to our current 'data'; note that in the next step you will write the logic for creating the newId number. | Hint: the current user 'data' is being imported via the 'loadData' function, and stored in a variable named 'data'  Hint: when pushing the newId and the newUser, remember that our 'data' elements should (in this assignment) be objects. They need to be in the form of key:value pairs - with keys that match those of our users.json |
| 10 | Write logic to create a new user ID number - one that will always be higher than the current highest id number in the array (so, if the current highest id # is "7", you will want to store "8" as the newId). Note that in order to do this, you will need to accomplish several things:  1. check the value of the id # of each element in the user 'data' array,  2. see whether it is greater than the previous element's id value,  3. store it if it \*is\* greater (and ignore it if it isn't), and  4. ultimately return (and store in your newId variable) a value that is one-greater than this highest-id value  While you can accomplish this logic in several ways - say, using a loop with a conditional - given what has been covered in this seciton of the sprint, you should try to use the l.reduce & Math.max methods | Hint: you will need to target your current user 'data' array with .reduce  Hint: you can use Math.max to evaluate which is greater - a currentValue, or a previousValue  Hint: you can use .reduce to store the result of Math.max - remember: it will store this result as the new previousValue  Hint: you can also tell .reduce to (for its first itteration) store "0" as its previousValue and (using Math.max) compare that to the zero'th element's id in your data array  Hint: make sure that you're adding 1 to what your .reduce method returns |
| 11 | Declare a new function - named "saveData" - that will 1. take in 'data', 2. use node's filesystem .writeFile method to 3. target our users.json file + converts our current 'data' (a JSON object) to a JSON string, and 4. return the result | Hint: use fs.writeFile  Hint: use JSON.stringify()  Hint: to understand why you need to use JSON.stringify, try logging 'data' by itself ... |
| 12 | Back in your 'run' function, invoke your 'saveData' function (at the very end of 'run's instructions), and have it take in your 'data'. Make it [a]wait for the process to finish. Log what is about to be saved to confirm things are working properly |  |
| 13 | Add a second conditional to 'run' that, if the 'operation' is "delete", will update 'data' to store what's returned from "deleteUser" - a function you will write in your next step. Pass 'data' and 'incomingData' into this function. |  |
| 14 | (After your 'addUser' function), declare a function named "deleteUser"; give it the parameters 'data' and 'removeId'. Make this function return a new version of 'data' - one that does not have the specified ID. | Hint: use .filter to check each element of the 'data' array  Hint: you want every element in 'data's array that \*does not\* have the id number of removeId  Hint: use '!=='  Hint: use parseInt to make removeId's number readable to the .filter method |
| ... |  |  |