Lab 2: Task List Server

Due Date: Before Lab on Week 3

Overview

In this lab, we'll be creating a task list application. This is the quintessential "Get Started" task in any web programming application. You'll be able to do the following operations on the task list:

1. Create a new task
2. Read all your tasks
3. Update the task
4. Delete any completed task

Lab 3 is dependent on Lab 2, so the best case scenario is that you get Lab 2 done within the week. Originally I'd created one longer lab, but by splitting it up, you get a chance to earn more points in the lab.

Objectives

1. Get familiarity with installing and using packages in Node.js
2. Create a CLI using Node.js
3. Perform CRUD operations on an Object in memory
4. Get familiarity with running the autograding.

Grading and Submission

This lab is autograded based on functionality. Run the test script using **npm run grade** command from your lab's directory. This should spit out a number which'll correspond to your grade.

Your submission should be a single zip file with everything in the lab-2-starter and any additional files EXCEPT the node_modules folder. DO NOT zip the node_modules folder.

Task 0: Read through the code

response.js

There's already plenty of code and comments written for you. Start in the response.js folder. We'll be using this for both Lab 2 and Lab 3, although it's not as useful for Lab 2.

Because JavaScript is not typed and you can add properties to objects at runtime, what value is returned if you forget to set a property somewhere and later access it in code

```

```

task-list.js

This is the bulk of this lab. We require (or import) both of our other files, sort-task-list.js, and response.js. Then we define an object called TaskListOrder with 2 Symbols. They could have been strings, but Symbols allow these values to be unique in our JavaScript program.

Next, we see a function similar to the responses.js which creates a Task for us given some parameters. Feel free to change around the parameters. One cool thing to note is the 3rd parameter, completed defaults to the value of false. That syntax with the equals sign for a parameter:

```
function (id, description, completed = false)
```

assigns the value of false to the completed parameter.

The big chunk of this function is the TaskList function. It has a taskCounter and an object containing the tasks. Each task will be added into the tasks object. Notice that inside of this function, we're creating more functions. This function creation within a function allows us to then return an object that interacts with local variables without ever exposing the local variables. If you're familiar with Object-Oriented Programming, this is one way to achieve encapsulation in JavaScript.

At the end of the TaskList function, we return the interface used to interact with a task list. This should seem pretty familiar from the Animal Shelter program in class.

```
return {
    createTask,
    completeTask,
};
```

It's done with a shorthand for creating objects. Because we already have a createTask variable (whose value is a function) and we want to create an object with the createTask key whose value is stored in the createTask variable, we can just use the name of the key and JavaScript will do the rest for us.

What value is exported outside of this file?

```
```

sort-task-list.js

Starting at the top, you have a require statement for the date-fns library. That should be a pretty big hint to use that. Next, we see the line

```
module.exports = { /** some code in here */ }
```

In Node.js (not JavaScript in general), module.exports allows us to export a value to use in other files. The value in this case is an object with 2 functions inside, but it could be a single number, a class, a function, any JavaScript value.

The first function to fill in is the sortByCreatedDate function. It takes in two Tasks and compares them, returning set values based on the documentation of Array.sort.

The second function to fill in is sortByCompletedDate. It's very similar to the sortByCreatedDate function, except that you have to deal with possible null values. Not every task is completed,

Task 1: Complete task-list.js functions

Each function in task-list.js is commented with what the function should do as well as the parameters to return. The suggested order for them is

1. createTask
2. readTask

3. completeTask
4. deleteTask
5. readAllTasks

The functions do what you think they should.

readAllTasks does have 1 weird-ish behavior. It takes in an order parameter. This order should be used to return the tasks in the specified TaskListOrder. Don't worry about completing this yet, you'll do that in Task 2.

This is a good point to make sure that you understand what you're doing. Ask the TA any questions about the code. Even if you "worked together" with someone, for your own sake and the next lab, make sure that you understand this code. The Create, Read, Update (complete), and Delete operations are the cornerstone of web programming.

Task 2: sort-task-list.js

Once you've filled in most of the task-list.js functions, move to sort-task-list.js. You have to fill in the two functions here. It's recommended to use the date-fns library to work with dates. Install that, add it to your package.json, code the functions, and move on with your life.

As mentioned above, make sure that you do some amount of null checking.

Task 3: Get a head start on Lab 3

Lab 2's handout was pretty detailed, but Lab 3's won't be as detailed. Use this time to go through it. It's a much more complex lab. Come to class on Thursday Week 3 with any questions you have about Lab 3.

Because Lab 3 has the solutions to this lab (in case you don't finish), Lab 3 will be locked until you submit this lab. You will get 1 submission attempt so make sure you're satisfied with your work.