Lab 3

Submit Assignment

Due Thursday by 11:59pm **Points** 100 **Submitting** a file upload

CS-546 Lab 3

Asynchronous Code, Files, and Promises

This week will be interesting -- we're going to be working with files; particularly, reading them, creating metrics on them, and storing them using promises.

You'll need to write two modules this week, and one test file.

Your file module, fileData.js

This module will export four methods:

getFileAsString(path)

This method will, when given a path, return a promise that resolves to a string with the contents of the files.

If no path is provided, it will return a rejected promise.

If there are any errors reading the file, the returned promise will reject rather than resolve, passing the error to the rejection callback.

getFileAsJSON(path)

This method will, when given a path, return a promise that resolves to a JavaScript object. You can use the `JSON.parse` function to convert a string to a JavaScript object (if it's valid!).

If no path is provided, it will return a rejected promise.

If there are any errors reading the file or parsing the file, the returned promise will reject rather than resolve, passing the error to the rejection callback.

Hint: this function can be accomplished in approximately 3-4 lines. Don't overcomplicate it!

saveStringToFile(path, text)

This method will take the `text` supplied, and store it in the file specified by `path`. The function should return a promise that will resolve to `true` when saving is completed.

If no path or text is provided, it will return a rejected promise.

If there are any errors writing the file, the returned promise will reject rather than resolve, passing the error to the rejection callback.

saveJSONToFile(path, obj)

This method will take the `obj` supplied and convert it into a string so that it may stored as in a file. The function should return a promise that will resolve to `true` when saving is completed.

If no path or obj is provided, it will return a rejected promise.

If there are any errors writing the file, the returned promise will reject rather than resolve, passing the error to the rejection callback.

Your metric module, textMetrics.js

Firstly, this module will export a method, `simplify(text)`. This method will take a string of text and will:

1. Convert the text to lowercase
2. Remove all non-alphanumeric characters (?.!'," and so on)
3. Convert all white space to simple spaces (new lines become spaces; tabs become spaces, etc)
4. Return the result.

Secondly, this module will export a method, `createMetrics(text)` which will scan through the text, simplify the text, and return an object with the following information based on the simplified text:

```
{
    totalLetters: total number of letters in the text,
    totalWords: total number of words in the text,
    uniqueWords: total number of unique words that appear in the text,
    longWords: number of words in the text that are 6 or more letters long,
    averageWordLength: the average number of letters in a word in the text,
    wordOccurrences: a dictionary of each word (lowercased, no punctuation) and how many times each word occurs in the text.
 }
```

So running:

`createMetrics("Hello, my friends! This is a great day to say hello.\n\n\tHello! 2 3 4 23")`

Will return:

```
    totalLetters: 49,
    totalWords: 16,
    uniqueWords: 14,
    longWords: 1,
    averageWordLength: 3.0625,
    wordOccurrences: {
        hello: 3,
        my: 1,
        friends: 1,
        this: 1,
        is: 1,
        a: 1,
        great: 1,
        day: 1,
        to: 1,
        say: 1,
        2: 1,
        3: 1,
        4: 1,
        23: 1
    }
}
```

app.js

Write a file, app.js, which will perform the following operation on each of these files (found in the Canvas Lecture 3 Module):

- chapter1.txt
- chapter2.txt
- chapter3.txt

For each file:

1. Check if a corresponding result file already exists for this file, if so query and print the result already stored.
2. If no result file is found, get the contents of the file using `getFileAsString`
3. Simplify the text, and store that text in a file named `fileName.debug.txt`
4. Run the text metrics, and store those metrics in `fileName.result.json`
5. Print the resulting metrics

So for example, with `chapter1.txt`, you will:

1. Check if `chapter1.result.json` exists; if it does, query and print the resulting object
2. If no result is found, perform `getFileAsString(chapter1.txt)`
3. simplify the text and store the result in `chapter1.debug.txt`
4. Run the text metrics and store those results in `chapter1.result.json`
5. Print the resulting metrics

General Requirements

1. You **must not submit** your node_modules folder
2. You **must remember** to save your dependencies to your package.json folder (if you use any)
3. You must do basic error checking in each function
    1. Check for arguments existing and of proper type.
    2. Throw if anything is out of bounds (ie, trying to perform an incalculable math operation or accessing data that does not exist)
    3. If a function should return a promise, instead of throwing you should return a rejected promise.