Projeto Classificatório

Processo seletivo - Web Development

Documentação

Hugo Mori

Sorocaba, 10 de agosto de 2021

Sumário

Documentação **3**

Questão 1 3

Questão 2 7

1. Documentação

Para resolução dos problemas foi realizada a implementação de um programa utilizando a linguagem JavaScript devido à facilidade para trabalhar com objetos, literais ou não, e sua fácil exportação para o formato pedido pelo exercício, o JSON, além de ser uma linguagem bem estruturada para tratamentos de requisições http e apresentar boas ferramentas para lidar com arquivos JSON. O funcionamento do programa se dá através de uma rotina procedural, onde por meio desta é realizado os procedimentos para correção do arquivo corrompido, assim como sua verificação ao final.
Portanto, o funcionamento do programa se dá, em ordem, através das chamadas das funções que serão descritas abaixo::

Observação:
Foi criada anteriormente algumas variáveis globais contendo o nome dos arquivos a serem abertos, dentre elas:

- *fileName*, contendo o nome do arquivo JSON "broken-database.json".
- *fixedFileName*, contendo o nome do arquivo JSON corrigido"saida.json".
- Um array de *produtos*, para a instanciação de objetos da classe Produto, criada para facilitar o recebimentos dos dados provenientes do JSON, assim como sua correção e gravação do arquivo corrigido.
- uma variável para para receber a abertura de requerimento de procedimentos com arquivos, *fs*.

1) Questão 1

a) Abertura do arquivo JSON corrompido

É realizado a chamada da função openFile(), a qual envia como parâmetro o nome do arquivo a ser aberto, e retorna um array de objetos abstraídos através do arquivo JSON.

```
// a) open broken database
var jsonFile = openFile(fileName)
```

```
// 1 a) Open File
function openFile(_file_name) {
    try {
        return JSON.parse(fs.readFileSync(_file_name, 'utf-8'))
    } catch (err) {
        // console.error(err)
        console.log(`\nOcorreu um erro na abertura do arquivo ${_file_name}.`)
        console.log(`Verifique se o arquivo existe ou está com o nome correto.`)
        console.log(`${err}\n`)
        return null
    }
}
```

A abertura do arquivo é realizado por meio da função readFileSync() - a escolha de um procedimento síncrono foi preciso para que o programa mantivesse sua ordem procedural -, que tem como parâmetros o nome do arquivo a ser aberto, e o *charset* do arquivo, no caso, como o arquivo pode conter assentos, foi utilizado o padrão UTF-8. Com o arquivo aberto, é feito a abstração dos objetos literais contidos nele através da função JSON.parse(). Esses objetos são então retornados, em forma de array, e armazenados na variável *jsonFile*.
Em caso de erro na abertura do arquivo, o erro é descrito e em seguida é retornado *null*, para informar que não houve sucesso e impedir que as demais etapas ocorram e gerem demais bugs.

b/c/d) Correções

Observação: Para realização da correção foi criado uma classe Produto a fim de receber os dados corrigidos do JSON para posteriormente serem exportados para um arquivo de forma mais fácil, através de um JSON.stringify() .

```
// ********************  CLASSES  ****************

class Produto {
    constructor(id, name, quantity, price, category) {
        this.id = id
        this.name = name
        this.quantity = quantity
        this.price = price
        this.category = category
    }
}
```

É realizado a chamada da função fixJSON(), a qual envia como parâmetro o array de dados extraídos do arquivo JSON, obtidos através da função openFile(), descrita acima.

```
// b) fix JSON and create objects produto to write repaired file
fixJSON(jsonFile)
```

A função fixJSON() é responsável por centralizar as funções de correções, e portanto realizar o procedimento de correção do arquivo corrompido. O procedimento é realizado através de um laço *for in* que varre a array de objetos extraídos do arquivo JSON, onde a cada iteração, é realizado a instanciado um objeto do tipo Produto, onde, nos parâmetros para seu construtor, são passado as chamadas de correções dos atributos corrompidos do arquivo JSON. Como a variável *produtos* é global, ela pode ser utilizada fora e dentro da função.

```
// 1 b/c/d) Fix JSON File and create objects produto
function fixJSON(_json_File) {
    // Create produtos objects
    try {

        const data = _json_File

        for (var i in data) {

            produtos[i] = new Produto(parseInt(data[i].id), fixString(data[i].name),
                fixQuantity(data[i].quantity), fixPrice(data[i].price), data[i].category)


        }

    }
    catch (err) {
        console.log(`${err}`)
    }
}
```

A função fixString(), que tem como parâmetro o atributo *name*, vindo do arquivo JSON, é corrigido através de um processo de varredura e substituição de caractere, onde, caso seja encontrado um dos caracteres corrompidos, é substituído pelo seu caractere original. Ao fim, é retornada a string corrigida.

```
// 1 b) Fix string function
function fixString(stringToFix) {
    stringToFix = stringToFix.replace(new RegExp(/æ/g), "a");
    stringToFix = stringToFix.replace(new RegExp(/¢/g), "c");
    stringToFix = stringToFix.replace(new RegExp(/ø/g), "o");
    stringToFix = stringToFix.replace(new RegExp(/ß/g), "b");

    return stringToFix
}
```

A função fixPrice() tem como parâmetro o atributo *price*, vindo do arquivo JSON, e é corrigido passando de string, para number, através da função parseFloat(). Ao fim do processo é retornado o atributo corrigido.

```
// 1 c) Fix prices
function fixPrice(priceToFix) {
    return parseFloat(priceToFix)
}
```

A função fixQuantity() tem como parâmetro o atributo *quantity*, vindo do arquivo JSON, a correção dele é feita verificando se ele não é um número, no caso, se ele não existe, caso se confirme, é retornado o valor 0, caso seja um número, é realizado um parseFloat() sobre o atributo, só por garantia, e retornado.

```
// 1 d) Fix quantities
function fixQuantity(quantityToFix) {
    let quantity

    if (isNaN(quantityToFix)) {
        quantity = 0
    }
    else {
        quantity = parseFloat(quantityToFix)
    }
    return quantity

}
```

e) Gravação do arquivo JSON corrigido

É realizado a chamada da função createAndWriteFixedJsonFile(), a qual envia como parâmetro o nome do arquivo JSON a ser salvo, a função é responsável por escrever os dados corrigidos.

```
// e) save new JSON file repaired
createAndWriteFixedJsonFile(fixedFileName)
```

A função createAndWriteFixedJsonFile() executa a tentativa de gravação do arquivo corrigido através da função writeFileSync - a escolha de um procedimento síncrono foi preciso para que o programa mantivesse sua ordem procedural correta - que tem como parâmetro o nome do arquivo a ser salvo e os dados que serão gravados. Como os dados estão em um array de objetos do tipo Produto, para transcrevemos ele para JSON utilizamos a função JSON.stringify(), onde nos parâmetros passamos os dados que serão formatados e o estilo de exibição, onde no caso queremos que não seja utilizado um estilo inline, e sim, que pule linhas.
Caso a tentativa falhe, o erro é informado e retornado null.

```
// 1 e) create and write fixed Json file
function createAndWriteFixedJsonFile(_fixedFileName) {
    try {
        fs.writeFileSync(_fixedFileName, JSON.stringify(produtos, null, 2))

    } catch (err) {
        console.log(`Ocorreu um erro na gravação do arquivo ${_file_name}.`)
        console.log(`Erro: ${err}\n`)
        return null
    }
}
```

2) Questão 2

a) Ordenação dos produtos, primeiramente, por categoria, em ordem alfabética, e por id, em ordem crescente.

A ordenação é realizada por meio da função sort(), que pode receber como parâmetro uma função de comparação, onde no caso, é passado a função order(), essa que recebe dois objetos para comparação. Os objetos são comparados pelo seu atributo *category*, do tipo string, usando, em caso de empate, o atributo *id*, do tipo number, para comparação.

```
// a) sort produtos for category and id
console.log(jsonFile.sort(order))
```

```
// 2 a)
function order(_produto_1, _produto_2) {
    if (_produto_1.category < _produto_2.category) {
        return -1
    } else if (_produto_1.category === _produto_2.category) {
        if (_produto_1.id < _produto_2.id) {
            return -1
        } else {
            return 0
        }
    } else {
        return 0
    }
}
```

b)Valor total do estoque por categoria

O cálculo do valor total do estoque por categoria é realizado através da função categorySumPrice(), que tem como parâmetro arquivo JSON recém arrumado. O funcionamento da função se dá por meio da criação de um objeto literal com os atributos *categoria* e *valor*, simbolizando, respectivamente, os atributos *category* e *price* do arquivo JSON, onde no caso, *valor* é o valor total do estoque, dado por meio da quantidade de produtos multiplicado pelo preço do produto. Em passos a função segue o processo abaixo:

1) É realizado um *for in* varrendo a array de objetos literais do arquivo JSON
2) Na primeira iteração, como ainda não houve a criação do objeto literal, somente é criado o objeto contendo os atributos *categoria* e *valor*, que recebem, respectivamente, os atributos *category* e a multiplicação de *quantity* e *price*, vindos do arquivo JSON. Esse objeto é guardado em um array com o nome *estoquePorCategoria*.
3) Nas demais iterações é realizado uma busca pelo valor contido no atributo *category*, do arquivo JSON, dentro do array *estoquePorCategoria*, caso este valor exista, é retornado o índice daquele objeto no array, por meio da função findIndex(), que recebe uma função callback como parâmetro, comparando as categorias. Caso não exista o valor, e consequentemente o índice, é retornado -1.
4) Caso o retorno do caso 3) seja diferente de -1, é realizado o acúmulo do valor de estoque no atributo *valor*, na array com a posição do índice retornado.
5) Caso o retorno do caso 3) seja -1, significando que ainda não houve a criação de um objeto com essa categoria, ele é criado, assim como na caso 2).
6) Ao fim do *for in* é realizado a impressão na tela do array *estoquePorCategoria*, listando as categorias e seus respectivos valores em estoque.

```
// b) Sum of the products price in the storage, by category
categorySumPrice(jsonFile)
```

```
// 2 b)
function categorySumPrice(_jsonFile) {
    let estoquePorCategoria = []
    let indice
    let countCategories = 0

    for (var i in _jsonFile) {
        if (i === 0) {
            estoquePorCategoria[countCategories] = { categoria: _jsonFile[i].category,
                valor: (_jsonFile[i].quantity * _jsonFile[i].price) }
            countCategories++
        }
        else {
            indice = estoquePorCategoria.findIndex(function(obj) {
                return obj.categoria == _jsonFile[i].category;
                });
            if (indice != -1) {
                estoquePorCategoria[indice].valor += (_jsonFile[i].quantity * _jsonFile[i].price)
            }
            else {
                estoquePorCategoria[countCategories] = { categoria: _jsonFile[i].category,
                    valor: (_jsonFile[i].quantity * _jsonFile[i].price) }
                countCategories++
            }
        }
    }

    console.log(estoquePorCategoria)
}
```