Projeto Classificatório

Processo seletivo - Analista em Tecnologia

Documentação

Larissa Severo de Proença

Março / 2022

Questões:

1. Recuperação dos dados originais do banco de dados

A - *Ler o arquivo Json;*

Realizei a leitura do arquivo **broken-database.json,** utilizei o método "**Require**" e nomeei minha função como "**readFile**"

O "try catch" é utilizado para tratar um erro caso ocorram problemas para fazer a leitura de um arquivo corrompido.

Optei pela criação de uma "let", pois iria utilizar o conteúdo da variável "data" apenas nesse bloco.

```
JS app.js > ...
1   const fs = require('fs')
2
3   function readFile(fileName) {
4       try {
5           let data = fs.readFileSync(fileName, 'utf-8')
6           return JSON.parse(data)
7       } catch (err) {
8           console.error(err)
9       }
10  }
```

Chamando a função:

```
91  const meuarquivo = readFile('/Users/raryo/rocky-challenge/resolucao.json')
```

Referência utilizada:

https://nodejs.dev/learn/reading-files-with-nodejs
https://www.geeksforgeeks.org/node-js-fs-readfilesync-method/

b) Corrigir Nomes;

Para a correção dos nomes criei a função '**fixNames**" e utilizei o "map" que realiza o retorno da função callback passada por argumento para cada elemento do Array, e assim, devolve um novo Array como resultado.

O método "**ReplaceAll**" foi essencial para exibir corretamente os caracteres incorretos. Preferi não utilizar o replace acompanhado do "g" (substituição de caractere global)

por acreditar que fica mais fácil de entender da forma abaixo:

A utilização de “const” foi necessária por se tratar de uma constante que não passará por uma nova atribuição de valor durante a execução do bloco.

```
13  function fixNames(data){
14      return data.map((el) => {
15          const fixedName = el.name.replaceAll("æ", "a").replaceAll("ø", "o").replaceAll("¢", "c").replaceAll("ß", "b")
16          return {
17              ...el,
18              name: fixedName
19          }
20      })
21  }
```

Referência:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/map
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/const

c) Corrigir Preço

Criei a função “**fixPrice**” para correção dos preços, utilizei dois “**return**” para que os valores fossem retornados devidamente.

Optei pela utilização do “**Number**” para transformar os valores em números.

```
23  function fixPrice(data){
24      return data.map((el) => {
25          return {
26              ...el,
27              price: Number(el.price)
28          }
29      })
30  }
```

Referencias Utilizadas:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Number

e) Corrigir quantidades;

Criei a função “**fixQuantity**” para corrigir os valores “**undefined**”

```
32  function fixQuantity(data){
33      return data.map((el) =>{
34          if (el.quantity == undefined){
35              return{
36                  ...el,
37                  quantity: 0
38              }
39          }
40          else{
41              return el
42          }
43      })
44  }
```

Foi necessário incrementar um “if” para realizar a verificação da existência de algum valor “**undefined”,** e caso existir, armazenar o valor 0.

Quando o valor está “**defined**” ele apenas retorna o valor original.

f) Exportar um arquivo JSON com o banco corrigido;

A função “SaveFile” realiza a exportação do arquivo através do WriteFile

```
43  function saveFile(fixedContent, destiny){
44      fs.writeFile(destiny, fixedContent, (error) =>{
45          if (error){
46              console.error(error)
47              return
48          }
49      })
50
51  }
```

Referência:
https://nodejs.org/en/knowledge/file-system/how-to-write-files-in-nodejs/

Em minhas pesquisas de aperfeiçoamento a utilização de funções em Javascript necessita se tornar uma prática e deve ser sempre utilizada.

2. Validação do banco de dados corrigido

a) **Uma função que imprime a lista com todos os nomes dos produtos, ordenados primeiro por categoria em ordem alfabética e ordenados por id em ordem crescente. Obs: é apenas uma saída, ordenada pelos dois fatores citados acima.**

A função “OrderItems” ficou responsável por realizar a ordem alfabética dos itens.

Novamente foi utilizado uma const
(*já que a variável não passará por uma nova atribuição de valor durante a execução do bloco.)*

O método “sort” ordena os elementos do próprio array e retorna os itens ordenados.

Primeiramente o filtro é feito pela ordenação da “category”, e depois pelo “ID”

```
56  function orderItems(data){
57      const sortedByID = data.sort((a, b) =>{
58          return a.id - b.id
59      })
60      return sortedByID.sort((a, b) => {
61          if (a.category > b.category){
62              return 1
63          }
64          if(a.id > a.id){
65              return -1
66          }
67          return 0
68      })
69  }
```

Referências utilizadas:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

b) **Uma função que calcula qual é o valor total do estoque por categoria, ou seja, a soma do valor de todos os produtos em estoque de cada categoria, considerando a quantidade de cada produto.**

Na função “sumTotalItms” foi necessário a criação de uma const com o nome das categorias.

Utilizei novamente o “map” para obter um novo Array como resultado optei pela utilização do método “**reduce**” para percorrer por todas os elementos e gerar um único valor.

O primeiro IF compara o valor atual que a categoria possui
=== (Valor e Tipo igual) – realiza a comparação entre duas variáveis, verificando o tipo de dados e comparando os dois valores.

O segundo IF verifica se há algum valor armazenado na categoria atual, caso não tenha, ele soma o valor acumulado.

Após é realizado a soma dos valores acumulados com a multiplicação do preço e quantidade para obter o valor final (soma do valor por categoria).

```
71  const categories = ['Panelas', 'Eletrodomésticos', 'Eletrônicos', 'Acessórios']
72
73  function sumTotalItems(items) {
74      return categories.map((category) => {
75        return {
76          category,
77          total: items.reduce((acc, current) => {
78            if (current.category === category) {
79              if (!current.quantity) {
80                return acc + current.price
81              }
82              return acc + (current.price * current.quantity)
83            }
84            return acc
85          }, 0)
86        }
87      })
88    }
89
```

Referências utilizadas:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

Chamada de funções:

```
91  const meuarquivo = readFile('/Users/raryo/rocky-challenge/resolucao.json')
92
93  console.log(Array.isArray(meuarquivo)) //verificar se é um array
94  console.log(fixNames(fixQuantity(fixPrice(meuarquivo)))) //funções para o save do json
95  console.log(sumTotalItems(orderItems(fixNames(fixQuantity(fixPrice(meuarquivo))))))
96  //executa as funções de soma e ordenação juntamente com os critéritos solicitados
97
98  const fixedFile = fixNames(meuarquivo)
99  saveFile(JSON.stringify(fixedFile), '/Users/raryo/rocky-challenge/saida.json')
```