Projeto Classificatório

Processo seletivo – TI

Estêvão Barros Chaves

Agosto de 2020

ROCKY
FULL DIGITAL PERFORMANCE

Sumário

Figuras

1. Explicação de cada funcionalidade

Funcionalidade 1 – Leitura do arquivo JSON

Foram utilizados os módulos ‘fs’ e ‘https’ do Node para a leitura do arquivo JSON [Figura 1].

Foi criada uma variável ‘file’ que recebe o nome do arquivo JSON a ser lido localmente (‘raw.txt’). Caso seja lido no servidor local através do método ‘fs.readFile’, os dados ‘rawData’ serão enviados para a função auxiliar ‘saveFile()’. Caso o arquivo não seja lido localmente, será chamada uma função ‘readFileServer()’ para buscar os dados JSON em um servidor web (‘https’) (disponível aqui), que serão enviados para ‘savefile()’, também através da variável ‘rawData’.

```
8   "use strict";
9   const fs = require('fs');
10  const https = require('https');
11  var file = 'raw.txt';
12  const url = "https://gitlab.com/-/snippets/1818996/raw";
13
14  // function1 - read JSON file (broken-database)
15  fs.readFile(file, (_err, rawData) => { //method to read file from local path
16      try {
17          saveFile(rawData);    //send raw data to auxiliar function
18          console.log('Successfully loaded from local server');
19      } catch (err) {
20          file = url; // sets file to the url constant
21          readFileServer(file); // calls this funtion if data from local path is not avaliable
22      };
23  });
24  function readFileServer(file){
25      https.get(file,(res) => { //gets data from web server
26          var rawData = ""; // variable will contain raw JSON data
27          res.on("data", (data) => {
28              rawData += data; // variable receive raw JSON data
29          });
30          res.on("end", () => {
31              saveFile(rawData); //send raw data to auxiliar function
32              console.log('Successfully loaded from https://gitlab.com/-/snippets/1818996/raw \t')
33          });
34          }).on("error", (error) => {
35              console.error(error.message);
36          });
37  };
38
39  // auxiliar function
40  function saveFile(rawData){
41      var data = JSON.parse(rawData); // variable containing the JSON raw data converted to JavaScript
42      //send js object to correction functions
43      correctName(data);
44      correctPrice(data);
45      correctQuantity(data);
46  }
```

Figura 1 – Recebe os dados JSON

Funcionalidade2 – Corrigir nomes

A função 'correctName()' recebe os dados e substitui caracteres errados por caracteres corretos. Para tal, o objeto é percorrido através do método 'forEach()' e, dentro deste, a lista 'wrongLetter' é percorrida, substituindo (método 'replace()') as letras erradas pelas letras corretas na lista 'rightletter' - através de seu index. Para tanto, é utilizada uma expressão regular (RegEx) que permite localizar globalmente os caracteres errados (linha 51) [Figura 2].

```
49  // function2 - Correct names
50  function correctName(data) {
51      const wrongLetter = [/ø/g, /æ/g, /¢/g, /ß/g] //affected letters
52      const rightLetter =  ['o', 'a', 'c', 'b'] //correct letters
53      //loop for each index inside data object
54      Object.keys(data).forEach(i => {
55          if(data[i].name.search('Wi-F') != -1){data[i].name += 'i'} //treats error exception Wi-F to Wi-fi.
56          //detect and replace globally wrong letters with RegEx, replacing bt the correct letters.
57          wrongLetter.forEach((letter, key) => {
58              data[i].name = data[i].name.replace(letter, rightLetter[key]);
59          });
60      });
61  };
```

Figura 2- Corrige nomes

Funcionalidade 3 – Corrigir preços

A função 'correctPrice()' percorre o index da lista, convertendo todos os valores da chave preço 'price' para 'Number()'.

```
63  // function3 - Correct prices
64  function correctPrice(data){
65      Object.keys(data).forEach(i => {
66          data[i].price = Number(data[i].price) // convert all prices in Number type
67      });
68  };
69
```

Figura 3 - Corrige preços

Funcionalidade 4 – Corrigir quantidades

A função 'correctQuantity()' [Figura 4] percorre o objeto 'data'. Em cada chave 'quantity', é utilizado um operador ternário. Caso 'quantity' não seja um número, ele recebe 0; caso já seja um número, ele é convertido em um número inteiro através do método 'parseInt()'; 'data' é convertido em formato JSON e enviado para a função 'writeFile()'.

```
70  //function4 - Correct quantities
71  function correctQuantity(data){
72      Object.keys(data).forEach(i => {
73          // if quantity not exists as a number, set equals to 0, else converts to Integer
74          isNaN(data[i].quantity) ? data[i].quantity = 0 : data[i].quantity = parseInt(data[i].quantity)
75      });
76          writeFile(JSON.stringify(data, null, '\t')); //convert js data to JSON data and sends to function writeFile()
77  };
78
```

Figura 4 - Corrige quantidades

Funcionalidade 5 – Exportar um arquivo JSON com o banco de dados corrigido

Uso da função assíncrona (ver Tratamento para evitar bugs) para exportar um arquivo JSON 'writeFile()' - recebe os dados já no formato JSON através da var 'data'. Utiliza o método fs.writeFile para gerar um arquivo 'saida.json. Ao fazer isso, a função recebe os dados ('data'), os converte para JS Object ('finalData') e os envia para as funções de validação [Figura 5].

```
79  //function 5 - Export JSON file with correct DataBase *** create or update 'saida.json'
80  async function writeFile(data){
81      fs.writeFile('saida.json', data,  (err) => {
82          if (err) throw err;
83          });
84          try {
85              var finalData = await JSON.parse(data)  //waits for receive the data and converts to be use
86              //send correct js object for validation
87              validationName(finalData);
88              validationQuantity(finalData);
89          } catch (error) {
90              console.error(error.message);
91      };
92  };
```

Figura 5 - Exporta arquivo JSON

Funcionalidade 6 – Validação: imprime nomes

A função ('validationName()') imprime os nomes dos produtos, ordenados por categoria(a-z) e, em seguida, por id(0-9) [Figura 6].

A função recebe os dados ('finalData') e, neste, através do método 'sort()', recebe uma arrow function com dois parâmetros de comparação. Utilizando uma operação ternária, compara as categorias 'category' e as retorna por ordem crescente (no caso String, [a-z]); caso encontre categorias iguais, organiza os 'id' por ordem crescente(no caso Number, [0-9]) (linha 99). Em seguida, a lista em 'finalData' é percorrida, adicionando ('push()') os nomes dos produtos na lista criada 'productList' que é impressa logo em seguida 'console.log' , convertida em formato JSON.

```
94  //***VALIDATION***
95  //function1(validation) - print the list of all names ordened alfabetically by category and crescently by id
96  function validationName(finalData){
97      //organize finalData object alfabetically by category and in case of equal categories by id
98      finalData.sort((a,b) => {
99          return a.category < b.category ? -1 : a.category > b.category ? 1 : 0 || (a.id - b.id)
100     });
101     var productList = [] // produtList will receive the product names
102     // add the organized names in the list productList
103     Object.keys(finalData).forEach(i => {
104         productList.push(finalData[i].name)
105     });
106     //print the product names by category and id in JSON format
107     console.log("Product names by category and id: ", JSON.stringify(productList, null, '\t'))
108 };
```

Figura 6 – Validação de nomes

Funcionalidade 7 – Validação: imprime valor do estoque por categoria

A função ‘validationQuantity()’ calcula o valor do estoque por categoria, considerando sua quantidade [Figura 7].

Foram criadas duas listas vazias (’categoryName’ e ‘valueByCategory’), onde são armazenados os nomes de cada categoria e o valor de estoque por categoria, respectivamente. Isso é feito através do método ‘forEach()’, que percorre o objeto: se o nome da categoria não for encontrado na lista ‘categoryName’, ele é lá adicionado *e* o valor do produto ‘price’ é multiplicado à quantidade em estoque ‘quantity’ e adicionado à lista ‘valueByCategory’. Caso a categoria já exista na lista ‘categoryName’, apenas o valor (‘price’ * ‘quantity’) da lista ‘valueByCategory’ é acrescentado (‘lastIndexOf()’).

Por fim, um objeto vazio ‘storage’ é criado. Percorre-se a lista ‘valueByCategory’ e para cada index, um valor lhe é atribuído (a partir da lista categoryName), utilizando uma arrow function (linha 128). Imprime o arquivo convertido em formato JSON.

```
108  //function2(validation) - calculate total value stock bt category and prints in JSON format.
109  function validationQuantity(finalData){
110      var categoryName = []  // list  receives the category names
111      var valueByCategory = [] // list receives the category values of stock
112
113      // checks if category name exists in categoryName list
114      Object.keys(finalData).forEach(i => {
115          // if category name does not exists in the list, add in the categoryName and the the total stock value (quantity * price)
116          if(!categoryName.includes(finalData[i].category)){
117              categoryName.push(finalData[i].category) &&
118              valueByCategory.push(finalData[i].quantity * finalData[i].price)
119          //  else, if category name already exists, increase the value of stock of that category (last index of the list category n
120          } else{
121              valueByCategory[categoryName.lastIndexOf(finalData[i].category)] += finalData[i].quantity * finalData[i].price
122          }
123      });
124      var storage = {};
125      // convert the two lists of category names and value of stock in a object
126      valueByCategory.forEach((key, i) => storage[categoryName[i]] = key);
127      // convert and print the object in JSON format
128      console.log("Storage price by category: ", JSON.stringify(storage, null, '\t'))
129  };
130
```

Figura 7 – Validação de preço/quantidade

2. O porquê da escolha da linguagem

A linguagem JavaScript foi escolhida devido ao contato e aos conhecimentos prévios em Node.js. Os módulos de leitura e exportação (‘fs’ e ‘https’) propiciaram o manuseio simples e eficiente dos arquivos de dados.

3. Tratamentos para evitar bugs

Foram utilizados alguns tratamentos para evitar bugs no código:

- uso do “use strict” no código, aumentando a restrição do código e, por conseguinte, sua segurança, sendo executado de forma mais rigorosa [Figura 1];
- uso da construção sintática ‘try(), catch()’ para lidar com erros Figura 1 e Figura 5);
- caso o arquivo JSON não seja encontrado localmente, é requerido via servidor web; e, caso o arquivo, web também não esteja disponível, responde com um erro (‘res.on(‘error’)’) [Figura 1 – Recebe os dados JSON];
- uso de função assíncrona, aguardando o recebimento dos dados exportados em arquivo JSON (saida.json). Garante que o código aguarde pelo carregamento dos dados para continuar sua execução. A depender do tempo requerido pelo sistema para ler os dados (tamanho do banco de dados), é útil para evitar que uma função de leitura dos dados seja chamada apenas após o carregamento destes; (Figura 5);

4. Considerações adicionais

- Na linha 55 [Figura 5Figura 6 – Validação de nomes], um if é utilizado para tratar um erro excepcional (adicionando a letra ‘i’ ao final do nome terminado em ‘Wi-F’), corrigindo-o para Wi-Fi.
- Para utilizar o ‘nodemon’, é necessário inserir o comando ‘nodemon --ext js resolucao.js’ para que o arquivo JSON não seja monitorado em looping (o que acarretaria a impressão em looping).