

Primeiramente gostaria de agradecer a oportunidade de fazer parte do processo seletivo da Rocky.

Vamos às considerações sobre o a resolução do teste classificatório:

-O ambiente de teste utilizado para verificar o funcionamento do arquivo `resolucao.js` criado durante o desafio foi uma distribuição Arch Linux(Garuda kernel 5.16-linuxzenmod) , portanto as barras utilizadas "/" são diferentes das utilizadas pelo MS Windows, mas tem o mesmo formato de arquivamento dos sistemas baseados em Unix como o MacOS, caso ocorra algum bug nos itens relacionados a escrita ou leitura, esta pode ser a causa;

- Na primeira função denominada **lerJson()** carrega o arquivo `broken-database.json` com a ajuda do `__dirname` (que visa fazer com o que o arquivo possa ser encontrado em diversos tipos de ambientes e sistemas, sem que haja a necessidade do "." antes do nome do arquivo) e o lê para que o processo de recuperação dos dados possa ser iniciado;

- A segunda função **fixDatabaseNames()** recebe o o `broken-database.json` que passou pela função **stringify()** para que os nomes dos produtos possam ser recuperados, dando origem ao **databaseNomesCorrigidos** e a sua versão em forma de objeto **parsedDatabaseNomesCorrigidos**, que através do uso do **globalThis** podem ser acessados fora do escopo da função, pois, como não é uma página web, não requer tanta preocupação com o escopo global e memória utilizada pela página, assim o objeto criado por esta função terá os caracteres corretos e poderá passar pelos outros processos de recuperação dos dados do banco;

- A terceira função **fixPrices()** se utiliza do método **parseFloat()** para transformar os preços que se tornaram strings durante a corrupção do banco de dados em números para posteriores funções que vão requerer este formato através da utilização da **Object.keys()** método específico para acessar e realizar mudanças em objetos na linguagem Javascript;

- A quarta função **fixQuantity()** verifica se existe o atributo **quantity** no objeto de id i, caso exista ele permanece com a mesma quantidade, caso não é realizado uma espécie de push no objeto, e ele passa a ter quantidade =0, "produtops" foi escrito de propósito;

- A quinta função **exportFixedDatabase()** apenas exporta os dados do atual objeto **parsedFixedDatabase** que passou pelas correções de nome, quantidade, preço passando antes pelo **JSON.stringify()** e utilizando como parâmetros o arquivo, `null(replacer)` e 2 que fará a indentação do arquivo ser de 2 espaços. A escrita foi feita utilizando o método **writeFileSync()** da biblioteca **filesystem(ou fs)** ;

- A sexta função **orderingByCategoryAndId()** recebe como input o arquivo **saida.json** criado na função anterior e o lê através do mesmo método utilizado na primeira função, o **readFileSync()** e partir disso imprimirá no console uma tabela com os itens ordenados primeiramente pela categoria em ordem alfabética e organizados por id crescente caso pertençam a mesma categoria, esta função como item adicional

também escreverá no arquivo **saida-ordenada.json** os mesmos itens em ordem idêntica a da lista gerada no terminal/console;

- A sétima e última função **calculateStockvalues()** também recebe o arquivo **saida.json** cuja leitura foi globalizada através do uso do **orderedParsedFixedDatabase** juntamente com o **globalThis**, e realizará a soma dos estoques de cada categoria de produtos, esta função também se utiliza do método **Object.keys** pois na sexta função o arquivo lido foi transformado novamente num objeto de modo a facilitar sua manipulação, para esta lista foi utilizada `` (crase ou "backticks" para que a saída de texto reconheça funções e variáveis javascript, foram utilizadas duas funções para facilitar a leitura dos valores obtidos, **toFixed()** para determinar a quantidade de casas decimais de cada resultado e **replace()** que substituiu os pontos do padrão americano para as vírgulas indicativas de decimal do padrão brasileiro, pois, como diria Dom Toretto: "Aqui é Brasil";

- Boa parte do tratamento de erros foi feita com blocos de **try** e **catch** de forma a indicar ao usuário qual das funções está apresentando algum problema, com exceção da função 1 que utiliza **if else** ;

- Os métodos **readFileSync** e **writeFileSync** foram utilizados pois, não necessitam da utilização dos parâmetros **err** e **result** dentro das callback functions, deixando mais claro e uniforme o código em geral;

- A biblioteca que permite a execução de todo o código é a **filesystem** ou **fs**, e portanto ela é necessária para o funcionamento do mesmo.

A escolha de javascript para linguagem neste desafio pode ser facilmente compreendida pelo fato de estar no mercado há mais de 25 anos e ainda recebendo atualizações constantes e diversas bibliotecas que tem como propósito facilitar a vida do usuário/programador, como jQuery, a filesystem(utilizada na resolucao.js), ejs(para html), dentre outras. Além de ter uma sintaxe mais simplificada que outras linguagens como java.

Por fim, vale destacar que neste projeto não foi realizado um rigoroso controle afim de evitar bugs, mas foi verificado contra erros de digitação e falta de coesão nos nomes das variáveis, objetos, arrays e outras estruturas "nomeáveis".