ROCKY

FULL DIGITAL PERFORMANCE

# Teste Prático

## Processo seletivo - Analista em Tecnologia

Higor Gonçalves Brandão

# O Problema

A situação-problema é dada a partir de um banco de dados que armazena as informações de produtos de um estoque em um arquivo JSON que sofreu alterações não desejadas, o que modificou o formato padrão do banco de dados, mas não causou perdas completas.

O objetivo do Teste Prático é recuperar os dados desses produtos para o formato padrão novamente, e, após isso, verificar se as correções foram feitas corretamente. O banco de dados possui 5 atributos que guardam as informações dos produtos: *id, name, quantity, price* e *category*.

Foram três problemas detectados no banco de dados corrompido: **nomes dos produtos com caracteres modificados, alguns preços com o tipo "string" ao invés de "number" e a ausência do atributo "quantity" em produtos que possuíam estoque 0.** Sendo assim, três funções foram criadas para a correção desses erros.

Após a correção do banco de dados, ele deveria ser exportado em um novo arquivo JSON, que será validado através de outras duas funções: **uma função que ordena os produtos primeiramente por categoria em ordem alfabética, e em seguida por id em ordem crescente; e uma função para calcular o valor total do estoque dos produtos de uma mesma categoria.**

Dessa forma, ao todo, foram criadas três funções para a correção e outras duas para validação do banco de dados corrigido, além das funções de leitura e escrita do arquivo, as quais serão mostradas abaixo.

# O algoritmo

1. **Recuperação dos dados originais do banco de dados**

   **1.1 - Leitura do arquivo: readDatabaseFile(database)**

   Para a leitura do arquivo broken-database.json corrompido, foi utilizada a biblioteca de manipulação de arquivos "FS". A função **readFileSync()** recebe o diretório do arquivo como parâmetro e cria uma string JSON. Em seguida, é utilizada a função **JSON.parse()**, que retorna um objeto descrito por essa string.

### 1.2 - Correção do nome dos produtos: fixProductName(product)

Nessa função, o atributo “name” de cada produto é acessado. Nele, será feito uma verificação que irá alterar todos os caracteres modificados no nome do produto. O método **replace** foi usado para essa correção. Ele receberá uma expressão regular como parâmetro com todos os caracteres inválidos e verificará globalmente a string de nome através da “flag” *g*. O segundo parâmetro do método irá realizar a troca dos caracteres corretos de acordo com o padrão que foi passado pela constante *characterFixer* criada anteriormente.

### 1.3 - Correção para tipo “number”: fixPriceType(product)

Dessa vez, o atributo a ser acessado será o “price”. É feito uma verificação no tipo de cada um dos atributos “price” de cada produto. Caso o tipo retornado seja “string”, é feito um tratamento através da função **parseFloat()**, que transformará a string em um ponto flutuante, e, consequentemente, mudando seu tipo para “number”.

### 1.4 - Correção do atributo de estoques: fixProductQuantity(product)

Para essa função, o atributo “quantity” de cada produto será verificado através da condição “if”. Caso a condição seja falsa, ou seja, não existe o atributo dentro do produto analisado, será criado um atributo de mesmo nome com valor 0.

### 1.5 - Exportação do arquivo corrigido: writeFixedDatabaseFile(database)

O banco de dados, agora no formato corrigido, deverá ser exportado para um novo arquivo .json, chamado “saida.json”. Para isso, mais uma vez será utilizado a biblioteca FS. A função para criar o arquivo será a **writeFile()**. Essa função, nesse caso, recebe três atributos: o nome do arquivo, armazenado na constante *outputFilePath*; a informação que será escrita no arquivo, que é passada através do JSON.stringify, com seus devidos parâmetros (dados do e configuração do espaçamento do arquivo); e a callback.

2. **Validação do banco de dados corrigido**

   **2.1 - Impressão da lista de produtos ordenados por categoria, em ordem alfabética, e id, em ordem crescente: printProductsInOrder(database)**

   Essa função receberá o banco de dados já corrigido e realizará, através do método **sort()**, a ordenação dos produtos por ordem alfabética de categoria. É feito uma verificação através de uma operação ternária, que compara o valor das categorias, ordenando-as de acordo com o retorno obtido. Após isso, será realizada uma nova ordenação, dessa vez ordenando o id do produto em ordem crescente. Será feita uma validação através de um “if” que verifica se a categoria de ambos produtos comparados é a mesma. Caso seja, uma nova operação ternária é feita, alterando a posição dos produtos, dessa vez apenas entre as categorias. No final, a lista é imprimida através do **console.log()**.

   **2.2 - Estoque dos produtos por categoria: totalStockValue(database)**

   Nessa função, inicialmente será criado um objeto vazio(*totalCategory*) que armazenará os nomes das categorias e a quantidade de produtos pertencentes a ela. Através do método **map()**, os elementos serão percorridos. Em cada um deles será feito uma verificação: caso não exista a categoria do produto verificado em questão dentro do objeto *totalCategory*, é criada essa categoria e ela recebe como valor de estoque inicial a quantidade do estoque deste produto. Caso já exista, a variável *total* irá receber o valor de momento do estoque da propriedade(“category”) analisada no momento. Em seguida essa propriedade receberá *total* + a quantidade do produto em questão. No final da execução do método, será impresso o objeto *totalCategory* preenchido com suas respectivas somas por categoria.

# Observações adicionais

Durante a implementação do algoritmo, foram feitas implementações para evitar bugs no código. A função de leitura do arquivo foi criada através do bloco **try catch**, o que proporciona uma maior segurança tanto para leitura e escrita quanto para o entendimento do erro, caso haja. A função de escrita possui um parâmetro de callback que identifica e mostra o erro caso haja algum durante a escrita do arquivo. Além disso, durante a implementação das funções, todas elas foram executadas de

forma isolada, de modo que seja mais fácil identificar e corrigir os erros que venham acontecer durante a execução da mesma.

A escolha da linguagem JavaScript, nesse caso, foi interessante pois o desafio envolve diversas iterações para sua resolução, e as ferramentas que a linguagem disponibiliza para isso são mais que suficientes, além de possuírem fácil implementação, mesmo àquelas as quais ainda não conhecia. Além disso, o Node.js consegue lidar facilmente com requisições para esse tipo de problema.

## Referências utilizadas


https://stackoverflow.com/questions/63708978/how-to-import-a-file-with-json-object-to-another-js-file - Leitura do arquivo

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

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String/replace - Replace

https://thispointer.com/javascript-replace-multiple-characters-in-string/ - Replace múltiplo

https://www.geeksforgeeks.org/node-js-fs-writefile-method/ - Escrita do arquivo