Projeto Classificatório

Processo seletivo - Web Development

Lucas Eduardo Pereira de Oliveira

Sorocaba

2022

Sumário

1. Introdução

O desafio solicita a criação de 5 funções para formatar e recuperar os dados de um banco de dados corrompido, também é solicitado duas funções para testar e validar os dados recuperados. Para cumprir o desafio foi feito um código em Javascript, que embora tenha sido obrigatório o uso, é a linguagem de programação que mais tenho estudado atualmente e seria a minha escolha mesmo sem a obrigatoriedade.

2. Resumo Algoritmo

O algoritmo que desenvolvi, chama a primeira função a *createObject()*, nela há chamada para as funções que concertam os nomes e preços (*recoverName()* e *recoverPrice()*), e então chama a função que cria o JSON concertado de saída (*saveJSON()*). Com o arquivo concertado chamo as funções de validação *validateList()* e *quantityPerCategory()*, que imprimem na tela respectivamente uma lista com os nomes ordenada por *categoria* e *id*, e uma sequência com a soma da quantidade em estoque de cada categoria.

3. Funções

3.1. readJSON(file)

Essa função recebe como parâmetro o caminho do arquivo que deseja abrir e retorna um vetor de objetos já sem o formato *JSON*, isso caso exista o arquivo, caso não exista ele retorna null.

3.2. recoverName(brokenDB)

Essa função que irá restaurar apenas os nomes, recebe como parâmetro o vetor de objetos que será concertado, então primeiro ele cria um vetor só com o campo nome dos objetos, que é percorrido e cada nome passa por uma sequência de funções *replace*, que substitui o valor errado pela letra correta, então armazena os nomes corretos em um vetor e retorna esse vetor.

3.3. recoverPrice(brokenDB)

Da mesma forma que a função anterior essa recebe o mesmo parâmetro, e cria um vetor apenas com os preços de cada objeto, então percorre esse vetor verificando se o valor não é um número, se não for ele converte para *float*, se ele já for um número ele apenas adiciona no vetor final que contém todos os preços em ordem com o tipo *float* e retorna esse vetor.

3.4. CreateObject()

Essa função é responsável por criar os objetos concertados, ele começa copiando os objetos com valores errados, chama as funções que retornam os valores de nomes e preços concertados, e então substituo os nomes e os preços pelos dos vetores, então verifico se o objeto tem o campo *quantity*, se ele não tiver então adiciono o campo com valor 0. Após isso temos um vetor com os objetos concertados, então é chamado a função que irá salvar num *JSON*.

3.5. saveJSON(data)

Essa função recebe como parâmetro o vetor de objetos, e tem como objetivo salvar esse vetor no arquivo *saída.json*, para isso ele começa aplicando a função *stringify* no vetor recebido, para obter em formato JSON, então uso a função *writFileSync*, para escrever no arquivo, caso de erro na escrita ele chama a *callback* criada na função *finished,* por fim chamo as funções de validação, embora elas dependem do arquivo saída ter sido criado, a função usada para escrever garante que as funções só serão chamadas após terminar de escrever no arquivo.

3.6. validateList()

Essa função depende da existência do arquivo *saída.json*, após ler esse arquivo e guardar o vetor de objetos, passo a função *sort*, que fara a ordenação em duas partes, primeiro ele verificara por categoria, caso a categoria seja a mesma então ele irá verificar por id, e ordenará esse id em ordem crescente, então é executado a função *map* do vetor ordenado para imprimir apenas o campo *nome*.

3.7. quantityPerCategory()

Essa função também exige a existência do arquivo *saída.json*, primeiro é criado um Set para adicionar as categorias presentes no arquivo saída, e por ser um set obrigatoriamente não terá repetições de categoria, então simplesmente passo por todo vetor de objetos adicionando o campo categoria nesse set, então para cada categoria é criado um vetor de objetos contendo todos os objetos daquela categoria, para isso utilizei a função *filter*, após isso é aplicado a função *map* para pegar apenas o campo *quantidade* desse vetor filtrado, e então é executado a função *reduce*, que irá somar as quantidades, e por fim é imprimido no console o resultado para cada categoria.

4. Conclusão

Durante o desenvolvimento tive dificuldade na hora de salvar o arquivo de saída, o qual utilizei como base o vídeo do canal *Charscript* (*https://youtu.be/T7s3st6xfpA*) e

com a questão da sincronia entre as funções, que durou até aprender o a função *writeFileSync*. Mas acredito que fora isso consegui entregar o que foi pedido, embora tenha ficado na dúvida sobre a execução da função de validação que soma as quantidades de uma categoria, afinal ainda não sei se deveria imprimir ou não na tela.