Rocky Full Digital Performance
Processo Seletivo - Web Development

Documentação do teste prático

Correção e ordenação de banco de dados corrompido

Gabriel da Silva Kyomen

Índice

1. Objetivos
2. Escolha da linguagem
3. Funcionalidades
    a. jsonReader
    b. jsonWriter
    c. fixDataNames
    d. fixDataPrices
    e. fixDataQuantity
    f. sortAndPrintData
    g. stockQuantityByCategory
4. Tratamento de erros
5. Considerações finais

Objetivos

A aplicação desenvolvida para esta fase do projeto classificatório visa a correção dos dados de um banco de dados *NoSQL*, erroneamente alterados durante a execução de uma rotina não testada. Para este fim, é necessário ler um arquivo `.json`, corrigindo os nomes, preços e quantidades dos produtos, que são os itens deste banco, guardando a solução em um novo arquivo `.json`. Também deseja-se validar a recuperação por meio da ordenação e contagem dos dados, imprimindo algumas informações na saída padrão.

Escolha da linguagem

Para construir a solução, foi escolhida a linguagem *JavaScript*, já que é o padrão para a manipulação de dados *JSON* (*JavaScript Object Notation*), além de já possuir conhecimentos prévios com a mesma.

Funcionalidades

1. jsonReader

    Recebe o caminho de um arquivo `.json`, faz a sua leitura síncrona utilizando o módulo `fs` e tenta convertê-lo em objeto. Retorna o objeto em caso de sucesso, 0 em caso de erro.

2. jsonWriter

    Recebe o caminho de um arquivo `.json` e um objeto *JSON* a ser gravado nele de forma síncrona utilizando o módulo `fs`. Sobrescreve o arquivo caso ele já exista. Retorna 1 em caso de sucesso, 0 em caso de erro.

3. fixDataNames

    Recebe o objeto *JSON* do banco de dados corrompido, corrigindo a propriedade `name` dos seus itens por meio da função `replace`, utilizando uma expressão regular para procurar todas as ocorrências dos caracteres errados, substituindo-os pelos esperados.

4. fixDataPrices

   Recebe o objeto *JSON* do banco de dados corrompido, verificando se a propriedade `price` dos seus itens é um texto por meio do operador `typeof`. Em caso afirmativo, converte-a para o tipo numérico, padrão do banco.

5. fixDataQuantity

   Recebe o objeto *JSON* do banco de dados corrompido, verificando se a propriedade `quantity` dos seus itens está presente por meio do método `hasOwnProperty`. Em caso negativo, insere-a no item com o valor 0.

6. sortAndPrintData

   Recebe o objeto *JSON* do banco de dados já corrigido, ordenando seus itens a partir da propriedade `category` em ordem alfabética, e, para itens de mesma categoria, ordena-os em ordem crescente de acordo com a propriedade `id`. Por fim, imprime a propriedade `name` de cada um dos itens.

   Ambos processos de ordenação são feitos ao passar para a função `sort` uma função anônima de comparação que recebe dois itens, retornando um valor negativo caso o primeiro deles deva ser colocado antes no vetor, o valor 0 caso sejam iguais e um valor positivo caso o segundo deles deva ser colocado antes no vetor. A categoria é pré-processada, colocando todos os seus caracteres em caixa alta, evitando que letras maiúsculas e minúsculas causem diferenças na ordenação.

7. stockQuantityByCategory

   Recebe o objeto *JSON* do banco de dados já corrigido e ordenado pela funcionalidade anterior, retornando um vetor contendo em cada posição o nome de uma categoria e a quantidade total de produtos desta mesma categoria.

   Começa criando um vetor vazio de categorias. Percorre então o JSON, e, para cada item dele, cria uma posição no vetor de categorias caso este seja o primeiro item do JSON ou caso a propriedade `category` do item atual seja diferente da mesma do item anterior. Independente da criação dessa posição ou da utilização da última posição existente, soma a quantidade do item atual na mais nova posição do vetor (maior índice criado). Ao fim, retorna o vetor de categorias criado.

Tratamento de erros

As funções de leitura e escrita de arquivos poderiam gerar erros caso o caminho para o arquivo passado como parâmetro não exista ou esteja incorreto. Para evitar que o algoritmo interrompa o funcionamento, foram adicionados blocos `try/catch` dentro delas, fazendo com que retornem 0 em caso de erro. Verificar o retorno dessas funções impede a manipulação dos dados, ou seja, no pior dos casos o banco de dados não será corrigido.

Considerações finais

Apesar de possuir conhecimentos prévios, parte da implementação trouxe novidades, permitindo novos aprendizados e a prática da habilidade de pesquisa e leitura de documentação, além de dar a oportunidade de discussão de alto nível com algumas pessoas da comunidade em torno da tecnologia utilizada.

Com isso, o desafio proposto trouxe grande evolução pessoal e profissional, incentivando ainda mais a continuar trilhando este caminho.