ROCKY - FULL DIGITAL PERFORMANCE

Projeto Classificatório

Este projeto foi elaborado como proposta de solução do problema do processo seletivo da Rocky – Full Digital Performance. O desafio consiste em recuperar dados corrompidos de um banco de dados, deixando-os no formato original e posteriormente realizar algumas validações nas correções.

A linguagem escolhida para realização do projeto foi Javascript, por ser a linguagem com a qual possuo maior familiaridade.

Requisitos:

- Ser capaz de ler o arquivo do banco de dados, no formato JSON.

- Percorrer o arquivo buscando os caracteres corrompidos e corrigi-los.

- Alterar o tipo de determinados valores de cada um dos objetos do arquivo corrompido.

- Adicionar propriedades excluídas em cada um dos objetos do arquivo corrompido.

- Exportar o banco de dados corrigido para um arquivo JSON.

- Ser capaz de organizar os dados de acordo com as exigências, e calcular o valor total em estoque dos produtos por categoria.

Solução:

Para resolução problema, utilizei o módulo File System, que possibilita a leitura e escrita de arquivos locais dentro do ambiente Javascript.

A solução consistiu na implementação de 6 funções, encontradas no arquivo ***resolucao.js***:

readFromFile(file) – Lê o arquivo e o retorna em formato JSON. Recebe a *string* **file,** contendo o diretório do arquivo com extensão (no caso, como o arquivo se encontra no diretório raiz, precisamos especificar somente o seu nome, isto é, 'raw.txt').

A leitura é feita através da função readFileSync(), do módulo *File System,* na qual foram passados os parâmetros **file** e **'utf-8',** no qual este último representa a opção de leitura escolhida.

fixData(data) – Corrige os caracteres corrompidos do arquivo **data**, em formato JSON.

Converte o *array* contido em **data** para uma *string* através do método JSON.stringify(). Após isso, percorre cada caractere desta string verificando se este é um dos caracteres corrompidos. Em caso

positivo, substitui o caractere corrompido pelo original. O mapeamento dos caracteres corrompidos e originais é feito através do objeto charModel.

fixPrices(data) – Recebe o *array* **data** e corrige o tipo dos valores da propriedade '*price'* dos objetos contidos no mesmo, mudando para o tipo *'number'* todos aqueles que não se encontram neste tipo.

Para isso, percorre cada um dos objetos verificando nestes se o valor da propriedade *'price'* não é um número. Em caso afirmativo, troca o tipo deste valor para *number*.

fixQuantities(data) – Recebe o *array* **data,** em formato JSON e corrige a ausência da propriedade '*quantity'* dentro de alguns objetos contidos no mesmo**.**

Isso é realizado através de uma varredura de cada objeto dentro do *array*, verificando através de um ternário a existência de um valor para a propriedade '*quantity'.* Caso esta não exista, cria a propriedade e seta seu valor para 0.

exportDataToFile(data, fileName) – Recebe o *array* **data** e o exporta para o arquivo de nome com extensão **fileName.**

A exportação para o arquivo é feita a partir da função writeFileSync(), do módulo *File System*, que recebe o nome de arquivo de saída e a formatação desejada, este último escolhido como uma string do tipo JSON.

sortData(data) – Função de validação do arquivo corrigido. Recebe o *array* **data** com os dados no formato JSON, já corrigidos, e imprime o nome dos produtos em ordem alfabética por categoria e em ordem crescente de id.

Para isso, percorre cada um dos objetos dentro de **data** criando a lista *sortedCategory*, contendo o nome de todas as categorias, sem repetições. Após sua criação, organiza os elementos de *sortedCategory* em ordem alfabética através do método *sort()*. Percorre então os elementos da lista *sortedCategory* e, para cada um destes, cria as listas *productsByCategory* e *sortedId*, sendo a primeira responsável por conter os objetos filtrados pela categoria em questão (usando o método *filter()* em conjunto com os elementos de *sortedCategory)* e a segunda por conter as propriedades '*id'* dos objetos em *productsByCategory*. Após realizar o ordenamento em ordem crescente dos elementos de *sortedId* pelo método *sort(),* utiliza cada valor de *id* desta lista para buscar os objetos de *productsByCategory,* imprimindo a propriedade *'name'* de cada um deles.

stockValue(data) – Segunda função de validação do arquivo corrigido. Recebe o *array* **data** com os dados no formato JSON, já corrigidos, e calcula o valor total em estoque dos produtos, por categoria.

Por questões de organização, a saída se encontra em ordem alfabética. Para isso, cria a lista *sortedCategory,* contendo a o nome das categorias, sem repetição e em ordem alfabética (ver a

explicação da função **sortData()**). Para cada uma destas categorias, cria a lista *productsByCategory*, contendo os objetos filtrados pelos elementos de *sortedCategory.* A partir disso, gera o objeto *categoryValue*, onde cada propriedade é uma categoria contida em *sortedCategory* e o valor corresponde a soma dos produtos das propriedades *'quantity'* e *'price'* de cada um dos objetos em *productsByCategory.* Ao final, imprime o objeto *categoryValue*.

execute() – Executa todas as funções anteriores, na ordem correta, imprimindo as saídas pedidas no console.

Testes unitários (opcional):

Para checagem das funcionalidades das principais funções, foi implementada uma rotina de testes unitários. Para se executar esta rotina (opcional), devemos instalar as dependências contidas no arquivo package.json através do comando:

- npm install

Assim, com o *jest* instalado, podemos verificar a cobertura dos testes através do comando:

- npm test.

A rotina será executada mostrado o resultado de testes de verificação de erros para principais funções de resolucao.js: **fixData(), fixPrices(), fixQuantities(), sortData()** e **stockValue().**