Processo Seletivo Rocky

Pedro Seabra Siqueira

Descrição dos arquivos

- **broken_db.json -** É o banco de dados de entrada usado para executar as funções.
- **saida.json -** É o banco de dados com os campos *name*, *price* e *quantity* corrigidos e padronizados (mas não ordenados).
- **resolucao.js -** Arquivo que contém a implementação e execução de todas as funções feitas.

Aspectos Gerais

Busquei utilizar boas práticas para nomear variáveis e funções, visando um código limpo e de fácil entendimento.

Apesar da entrada do Teste Prático ter uma estrutura previamente conhecida, desenvolvi as funções buscando uma maior generalização para conseguir lidar com entradas diferentes e melhorar a reusabilidade dessas funções.

Busquei também usar da melhor maneira os recursos facilitadores oferecidos pela linguagem, como a atribuição por desestruturação de objetos e os métodos existentes para trabalhar com arrays.

Optei por exibir as informações de maneira simples, no console, e para que a visualização fique melhor recomendo deixar o terminal em tela cheia.

Escolha da linguagem

Dentre as opções era a única com a qual eu já havia trabalhado, além de estar estudando um pouco de nodeJs recentemente, o que foi mais um incentivo para a escolha.

Uma outra vantagem do javascript é a facilidade do mesmo em trabalhar com o formato JSON, e funções como *sort* e *reduce* que facilitam bastante a utilização de arrays.

Tratamento de erros

Todas as funções implementadas utilizam blocos try-catch para capturar exceptions e exibir uma mensagem de erro no console do usuário.

Além da captura de exceptions lançadas pelo sistema, todas as funções verificam tipos de variáveis e propriedades de objetos para garantir que não gerem resultados inesperados. Por exemplo, a função *fixPrice* após converter o valor para float verifica se o novo valor é um número e lança uma exception caso não seja.

Inicialmente minha ideia era que o lançamento de exceptions utilizasse uma instância da classe Error do javascript, mas o construtor da classe só permite a definição da mensagem e não do nome do erro, dificultando a identificação de erros diferentes. Por isso foi criada uma função auxiliar que cria esses objetos

com o par de propriedades nome e mensagem (que também estão presentes nas exceptions específicas do javascript como TypeError, ReferenceError, etc).

Funções Auxiliares

newError

Retorna um objeto de erro com as propriedades nome e mensagem. Utilizado para lançar exceptions quando tipos inesperados são encontrados na execução do programa.

errorMessage

Recebe um objeto de erro (criado pela função newError ou pelo lançamento de alguma exception do sistema) e acessa suas propriedades por desestruturação do objeto, para então exibir uma mensagem no console do usuário.

Explicação das funções

loadFile

Recebe o caminho para o arquivo de entrada e o carrega.

saveFile

Salva o arquivo de saída. Recebe o nome, o conteúdo e a extensão desejada (opcional). Aceita outros formatos além de json, mas se a extensão não for informada json é o formato padrão.

fixText

Altera os caracteres errados no texto. Recebe o texto a ser alterado (a propriedade 'name' de cada objeto do arquivo de entrada) e um ou mais objetos com as seguintes propriedades: find e change. Para cada par, todas as ocorrências de 'find' na string de entrada serão substituídas pelo valor de 'change'.
Caso o conjunto de regras seja omitido, utiliza as regras informadas na descrição do Teste Prático e definidas no código na constante *nameRules*.

fixPrice

Corrige números que estão como tipo string para tipo number. Recebe o valor a ser convertido e utiliza a função parseFloat para obter, e em seguida retornar, a versão numérica, caso não seja um valor numérico uma exception é lançada.

fixQuantity

Corrige os itens que perderam a propriedade *quantity*. Caso o objeto já tenha a propriedade *quantity* a função retorna sem alterações, do contrário ele adiciona a propriedade ao objeto com valor 0.

fixOrder

Ordena o array de produtos crescentemente por categoria e id. Recebe a array a ser ordenada, e utiliza a função sort para fazer a ordenação.

A função sort percorre o array em pares de itens (*a* e *b*). Em cada comparação, caso o valor retornado seja maior que 0, o item *b* vem antes do item *a*; caso o retorno seja menor do que 0, o item *a* vem antes do item *b*; caso o retorno seja 0, a ordem dos itens não é alterada.

Para ordenar strings e números basta uma simples comparação de maior/menor que (ex: *a > b, b < a*, etc). Nesse caso, primeiro comparamos as propriedades *category* e *id* dos itens. Caso a categoria de *a* seja maior que a categoria de *b* (*a > b*), retornamos 1, caso seja menor retornamos -1, e caso sejam iguais verificamos a comparação entre os *ids*.

A função não tem retorno, alterando a própria array de entrada durante sua execução e imprimindo a nova ordenação no console (em formato de tabela para melhor visualização).

getTotalValueByCategory

Utilizando a função reduce na array de produtos, retorna o valor total em estoque agrupado por categoria. A função recebe dois parâmetros: uma função callback a ser aplicada em todos os itens do array, que por sua vez recebe os parâmetros *resultArr* (que é o acumulador, uma variável que recebe o retorno da callback e repassa o seu valor atualizado para cada iteração subsequente pelo array. Nesse caso, o acumulador é um array), e o segundo parâmetro é o valor inicial do acumulador, ou seja, de resultArr, que nesse caso é um array vazio (para retornar um array contendo apenas as informações de categoria e valor total em estoque).

A cada iteração, nossa callback verifica se a categoria do item atual é uma chave existente em *resultArr*. Se ela não existir, é a primeira vez que esta categoria apareceu e portanto ela é adicionada como uma chave em resultArr, com o valor total (preço x quantidade) em estoque daquele produto. Caso a chave exista, o valor anterior é somado ao valor total do produto atual.

Ao fim da execução os valores totais em estoque por categoria são exibidos no console, em formato de tabela.

index

Utilizada para executar as outras funções e exibir os resultados.

Primeiro faz o carregamento do arquivo de entrada (*broken_db.json*) usando a função *loadFile*. Caso o carregamento seja bem sucedido, verifica se é uma array, exibindo uma mensagem de erro caso não seja. Se for array, executa um loop foreach, e corrige os campos *name*, *price* e *quantity* de cada elemento.

Após a correção dos dados, tenta salvar o array atualizado em um arquivo json de saída. Caso a gravação do arquivo seja bem sucedida, carrega o mesmo e exibe os elementos ordenados por categoria e id e depois exibe o valor total em estoque por categoria de produto.