DOCUMENTAÇÃO DO TESTE PRÁTICO

Processo Seletivo Rocky

Nome: Arthur Felipe Bravo Pita

Data de Entrega: 10/08/2021

Contextualização

O presente relatório tem como objetivo documentar e explicar o algoritmo desenvolvido para a etapa do Teste Prático do Processo Seletivo de TI da empresa Rocky, área de Web Analytics. A situação problema consistia em um banco de dados corrompido que deveria ter seus dados devidamente recuperados, adequados ao formato correto e, por fim, validados.

Introdução

A atuação se deu a partir de um arquivo JSON fornecido (*broken-database.json*), documento onde estavam armazenados os dados de produtos. Seu tratamento (transformações e verificações) foi feito utilizando-se JavaScript, no arquivo *resolução.js*. Este, por sua vez, gerou o novo arquivo JSON corrigido, denominado *saida.json*.

Como nenhuma dessas tecnologias e linguagens era muito conhecida por mim até então, todo o desenvolvimento se deu com embasamento teórico em pesquisas na internet, tudo referenciado ao final dessa documentação (seção de Referências).

Explicação das funcionalidades

Funções criadas para a primeira questão, de recuperação dos dados originais do banco:

- **`readJson(path)`**

Função de leitura do arquivo JSON. Ela recebe em seu único parâmetro "**`path`**" o caminho e respectivo nome do arquivo a ser lido – isso torna possível reutilizá-la para leitura do arquivo corrigido gerado, posteriormente. Em seu interior, ela usa o método "**`fs.readFileSync`**" [1] para ler e atribuir a uma variável temporária o conteúdo do arquivo externo e também "**`JSON.parse`**" [2] para processar esse conteúdo – transformar em um objeto JavaScript.

Observação: foi escolhido o método síncrono "**`readFileSync`**" (ao invés do assíncrono "**`readFile`**"), pois, além de garantir uma eficiência um pouco maior, ele também promove certa segurança quanto a execução geral do programa (só partimos para a próxima instrução quando a leitura já foi concretizada). Como a maneira síncrona não inclui função de call-back, o tratamento foi feito com bloco "**`try/catch`**", mais detalhado adiante.

- **`correctName(toCorrect)`**

Função de correção dos nomes de produtos corrompidos. Seu único parâmetro, "**`toCorrect`**", é o conteúdo do arquivo JSON lido na função anterior. Com isso, a função utiliza um loop "**`for...in`**" [3] para percorrer todo o vetor de produtos e fazer as devidas substituições de caracteres modificados, recuperando "**a**" no lugar de "**æ**", "**c**" no de "**¢**", "**o**" no de "**ø**" e "**b**" no de "**ß**" dos nomes originais. Para essas trocas, foi utilizado o método "**`replace(x, y)`**" [4], que substitui o valor de seu parâmetro "**`x`**" pelo de "**`y`**". Também foi utilizado o sinalizador "**`g`**" (***global***) para incluir todas as ocorrências especificadas, e não só a primeira.

- **`correctPrice(toCorrect)`**

Nesta função de correção dos preços com tipo errado (***string***), também há apenas um parâmetro: "**`toCorrect`**", o objeto JavaScript com o conteúdo do JSON. Por sua vez, seu loop "**`for...in`**" verifica se os preços do vetor de produtos são de um tipo que não o ***number***. Se sim, os corrige com o método "**`parseFloat`**" [5], uma simples conversão para ***float*** (um tipo de número).

- **`correctQuantity(toCorrect)`**

Última função de correção dos dados corrompidos, essa tem o objetivo de ajustar os produtos em que o atributo de quantidade sumiu para os casos nulos. Para tanto, seu loop "**`for...in`**" percorre o parâmetro de vetor de produtos "**`toCorrect`**" em busca de atributos "**`quantity`**" faltantes – para isso, é usado o método de checagem de existência "**`hasOwnProperty`**" [6]. Caso aponte que a propriedade realmente não existe para aquele produto, ela é criada com o seu valor original: 0.

- **`writeJson(path, data)`**

Última função da questão 1, que tem por objetivo salvar todas as correções ao exportar num novo arquivo JSON com a saída corrigida. Para isso, possui dois parâmetros: "**`path`**" com o caminho e respectivo nome do arquivo que será criado e "**`data`**" com os dados a serem escritos no mesmo (nosso objeto JavaScript). Através do método "**`JSON.stringify`**" [7], esse objeto JavaScript é convertido para uma ***string***. Agora no formato certo, essa ***string*** é salva num arquivo JSON com o método "**`fs.writeFileSync`**" [8], que recebe como parâmetros o caminho em "**`path`**" e a própria ***string***.

Observação: assim como na leitura do arquivo, foi escolhido o método síncrono "**`writeFileSync`**" ao invés do assíncrono "**`writeFile`**" dada a melhor eficiência e segurança na continuação da execução do programa.

Funções criadas para segunda questão, de validação do banco de dados corrigido:

- **`sortProducts(toSort)`**

Função de ordenação e impressão de todos os nomes dos produtos do banco de dados corrigido. Ela contém apenas um parâmetro, o objeto JavaScript pós questão 1, ou seja, já corrigido. Para começar, é feita a ordenação das categorias por ordem alfabética utilizando-se do método "**`sort`**" [9] e uma função de comparação auxiliar que especifica como se dará o procedimento – se o seu resultado é menor que 0, o seu primeiro parâmetro é ordenado na frente do segundo; se for 0, não altera as posições entre si; se maior que 0, ordena o segundo parâmetro na frente do primeiro. Para evitar confusões entre maiúsculas e minúsculas, os mencionados “primeiro” e “segundo” parâmetros são definidos apenas em letras minúsculas através do método "**`toLowerCase`**" [10].

A segunda etapa da função consiste em ordenar os ID’s em ordem crescente. De modo semelhante à ordenação anterior, junto do método "**`sort`**" é definida uma função de comparação auxiliar para tratar de números (se isso não fosse feito, o primeiro Unicode do número definiria sua posição e não o seu valor na íntegra). Para esse caso, a auxiliar é baseada na subtração do ID do primeiro parâmetro pelo do segundo, tendo resultados idênticos a da anterior (resultado positivo ordena o primeiro na frente e assim por diante...).

A terceira e última etapa é percorrer e imprimir os nomes de todos os produtos do vetor após essas ordenações por categoria e ID realizadas. Para isso é usado um loop "**`for...in`**".

- **`countStock(toCount)`**

Por fim, a última função do programa consiste no cálculo do valor total em estoque de cada categoria de produtos. Ela também recebe como parâmetro e trata do objeto JavaScript corrigido e, agora, ordenado. Sua execução gira em torno de duas possibilidades: o produto do índice atual ter a mesma categoria que o próximo ou não. Assim, um loop "**`for`**" simples percorre desde a primeira posição até a penúltima do vetor de produtos ordenados, sempre

fazendo essa verificação. Caso a categoria atual seja diferente da próxima, ou seja, estamos tratando do último produto desta categoria, somamos o seu valor multiplicado pela quantidade junto aos anteriores (ou a 0, caso seja o primeiro) e imprimimos associado ao nome da categoria. Caso se trate do segundo caso, onde a categoria atual é igual a próxima, significa que ainda estamos somando uma mesma categoria, então apenas adicionamos a multiplicação do valor pela quantidade desse produto aos outros e partimos para o próximo.

Como era sempre feita uma verificação entre o índice atual e o seguinte, só era possível chegar até a penúltima posição, senão tentaríamos acessar uma fora do vetor. Logo, restou a última execução do loop, do último produto do vetor. Essa é feita do mesmo modo que todas as anteriores, finalizando a contagem e impressão de todas as categorias.

Tratamentos feitos no código para evitar bugs

Os erros mais possíveis de ocorrerem nesse código eram relativos à leitura e escrita de um arquivo externo. Portanto, na função "**`readJson`**" foi utilizado um bloco "**`try/catch`**" para tratamento de exceções. O mesmo envolveu o trecho com o método "**`fs.readFileSync`**" e o tratou com uma mensagem de erro indicando a impossibilidade de leitura.

Algo semelhante foi feito na função "**`writeJson`**", só que agora com o bloco "**`try/catch`**" englobando o método "**`fs.writeFileSync`**", para o tratamento de exceções de criação e escrita de arquivos.

Conclusão

De início tive certa preocupação por não ter tanto conhecimento acerca dos assuntos propostos (NoSQL, JSON, JavaScript e afins). Contudo, no decorrer do desenvolvimento e pesquisas, pude perceber o quão ampla a linguagem JavaScript pode ser, suas diversas aplicações e bibliotecas para os mais diversos fins. O mesmo com JSON, que eu nem imaginava que era utilizado em tantos lugares e de maneira tão simples.

Foi ótimo acrescentar mais esses aprendizados ao meu currículo, espero poder vencer novos desafios em breve, desta vez vestindo a camiseta da Rocky!

Link do Repositório no GitHub

https://github.com/Arth-Felipe/Rocky2021_TestePratico.git

Referências

A seguir, os links de embasamento para maior conhecimento sobre cada um dos tópicos descritos. Nenhuma cópia direta de execução foi feita, apenas exemplos parciais e explicação teórica foram aproveitadas para melhor entendimento.

Introdução no assunto

- JavaScript basics - Developer Mozilla
- Aprenda JSON em 20 minutos - Matheus Battisti (YouTube)
- Instalar o Node.js no WSL2 - Microsoft

[1] "fs.readFileSync" e [2] "JSON.parse"

- JavaScript object basics - Developer Mozilla
- Como Ler Arquivos Json com Nodejs - Medium
- Ler e gravar arquivos com File System - Filipe Deschamps (YouTube)

[3] "for...in"

- for...in - Developer Mozilla
- JavaScript: for, for...in, for...of - DevMedia

[4] "replace(x, y)"

- String.prototype.replace() - Developer Mozilla
- String replace() para substituir substrings no JavaScript - Medium

[5] "parseFloat"

- parseInt() - Developer Mozilla
- parseFloat() - Developer Mozilla

[6] "hasOwnProperty"

- Object.prototype.hasOwnProperty() - Developer Mozilla

[7] "JSON.stringify" e [8] "fs.writeFileSync"

- Ler e gravar arquivos com File System - Filipe Deschamps (YouTube)
- What is the difference between fs.writeFile and fs.writeFileSync - Medium

[9] "sort"

- Array.prototype.sort() - Developer Mozilla

[10] "toLowerCase"

- String.prototype.toLowerCase() - Developer Mozilla