Projeto Classificatório

Mateus Junio da Silva Miranda

30 de Março de 2022 – Sorocaba, SP

Descrição geral do projeto

Este algoritmo foi elaborado com o intuito de corrigir o banco de dados corrompido que se encontra no formato de objeto Json. Para manipular e corrigir, foi utilizada a linguagem front-end JavaScript. De maneira geral, foi importado o arquivo 'broken-database.json', manipulado de acordo com as correções necessárias e, por fim, exportado o arquivo 'saida.json', que devolve o banco de dados corrigido e modificado conforme o esperado.

Explicação passo-a-passo das funções criadas

De início, foi utilizado o **require** para que houvesse a inicialização dos arquivos de sistema e busca do banco de dados antigo:

```
// // Inicialização de arquivo de sistema
const fs = require('fs');

// // Realizando a localização do banco de dados através do método require
var jsonBanco = require('./broken-database.json');
```

Depois foram acionadas todas as funções criadas no escopo do código, observe:

```
//Chamando todas as funções de correção
for(i = 0; i < jsonBanco.length; i++){
    corrigirLetras(i);
}
corrigirQuantidade(jsonBanco);
corrigirPrecos(jsonBanco);
corrigirOrdem(jsonBanco);
```

Após ler o arquivo com o banco de dados corrompido, chamar todas as funções de correção e realizar as alterações e ordenações necessárias, o banco de dados foi exportado:

```
//Exportando o banco de dados corrigido
exportarJson(JSON.stringify(jsonBanco));
```

Em seguida foi chamada a função de soma por categoria e, para que fosse possível visualizar, o objeto Json foi convertido em string, através do método **JSON.stringify()**:

```
//Chamando função de soma por categoria
console.log(somarItens(jsonBanco));

//Exibindo função de soma por categoria
console.log(JSON.stringify(somarItens(jsonBanco)));
```

Funções criadas no decorrer do projeto

corrigirLetras

Para alterar os caracteres corrompidos, foi desenvolvida a função **corrigirLetras**. Nela foi utilizado o método **replaceAll()** para que todos os caracteres corrompidos da posição *name* pudessem ser substituídos:

```
// Função para modificar letras/caracteres corrompidos
function corrigirLetras(i) {
    jsonBanco[i].name = jsonBanco[i].name.replaceAll('æ', 'a');
    jsonBanco[i].name = jsonBanco[i].name.replaceAll('¢', 'c');
    jsonBanco[i].name = jsonBanco[i].name.replaceAll('ø', 'o');
    jsonBanco[i].name = jsonBanco[i].name.replaceAll('ß', 'b');
    return(jsonBanco[i].name)
}
```

corrigirPreco

Para corrigir o preço, foi necessário utilizar iniciar uma variável de posição element no objeto e converte-la para tipo number, através do método **parseFloat()**:

```
//Função que corrige os números que possuem valor string, para number
function corrigirPrecos(i) {

    for(var element in i) {
        var priceErr = i[element].price
        priceErr = parseFloat(priceErr)

        i[element].price = priceErr
    }
    return i;

}
```

corrigirQuantidade.

Para realizar a inserção da quantidade igualada a zero, nos itens em que a posição não existia, foi criada a função **corrigirQuantidade**. Para isso, houve a utilização do método **map,** para percorrer todos os objetos e posteriormente fazer a inserção do “quantity = 0” nos itens necessários:

```
// Função para corrigir a quantidade 0 apagada
function corrigirQuantidade(i){

        return i.map((pos) =>{
            if (pos.quantity == undefined){
                pos.quantity = 0;
            }
            else {
                return pos;
            }
        })
}
```

corrigirOrdem

Para ordenar os itens de maneira categórica e por id, foi desenvolvida a função **corrigirOrdem**. Nela foi utilizado o método **sort** para comparar os valores e ordená-los sequencialmente. Caso fossem da mesma categoria:

```
// Iniciando função de ordenação por categoria ou id
//obs: A fução abaixo faz referência ao seguinte link: https://goma
function corrigirOrdem(i)
{
    i.sort(function ( a, b){
        if(a.category < b.category) {
            return -1;
        }
        else
        {
            if ((a.category == b.category) && (a.id < b.id))
            {
                return -1;
            }
            return true;
        }
    });
    console.log(i);
    return i;
}
```

somarItens

Para somar os itens do estoque por categoria, foi criada a função **somarItens**, que declarava três variáveis; uma para guardar os valores por categoria; uma para iterar os valores; outra para guardar na posição ‘a’ da estrutura, o valor antigo. No caso, se o banco já está ordenado, a variável que guarda o valor anterior apenas identifica se o objeto possuí nesta categoria e, insere a categoria através do método **push**:

```
//Função para somar a quantidade dos itens, de acordo com a categoria
function somarItens(i) {
    var itens = [];
    var itera = -1;
    var antCategory = '';
    for(a in i) {
        if(antCategory != i[a].category) {
            itens.push({
                'category': i[a].category,
                'quantity': 0
            })
            itera++;
        }

        itens[itera].quantity += i[a].quantity;
        antCategory = i[a].category;
    }

    return itens;
}
```

exportarJson

Nesta função foi utilizado o **fs.appendFile()** para exportar o arquivo e caso esta exceção ocorresse, a mensagem de “Arquivo Salvo!” ocorre:

```
//Função para exportar o banco de dados
//obs: A função abaixo faz referências ao seguinte link: https
function exportarJson(i)
{
    fs.appendFile('saida.json', i, 'utf8', function (err) {
        if (err) throw err;
        console.log('Arquivo Salvo!');
      });
}
```

Descrição dos métodos, funções e instruções utilizadas

Require → Uma instrução node. js embutida, utilizada para incluir módulos de outros arquivos e ler o JS executando e retornando o objeto exportado;

JSON.stringify → Método utilizada para converter o objeto ou valor JSON para string;

replaceAll → Método que traz um retorno de string com todas ocorrências estabelecidas, modificadas pela nova substituição;

parseFloat → Uma função que recebe um argumento de entrada e, se for o caso, converte este argumento do tipo string em ponto flutuante number;

map → Método utilizado para percorrer toda uma estrutura de array, executar uma determinada ação e retornar o novo conteúdo;

sort → Este método classifica os elementos de acordo com valor retornado na função compare(), onde o menor valor retornará na ordem inicial

push → Serve para adicionar um novo valor a posição final de um array, retornando seu novo comprimento;

fsAppendFyle → Método utilizado para anexar de forma assíncrona os dados de um arquivo e até mesmo criar, caso não exista.