Grupo Raccoon

Documentação

Teste TI: Web Development

Sabrina Veloso
7/1/2022

O Problema

Você é responsável por um software de gestão de estoque de produtos. Ao fazer uma alteração no sistema, uma rotina que não foi devidamente testada acabou quebrando todo o banco de dados. Por sorte, não houve perda completa dos dados, mas eles não estão mais no formato esperado pelo sistema. Sua missão nesse projeto é recuperar os dados e deixá-los no formato adequado novamente. Além disso, você precisará criar também alguns métodos para validação das correções.

O banco de dados utilizado é um banco de dados NoSQL, orientado a documentos. Não se assuste caso você não conheça esses nomes. Não iremos mexer diretamente com banco de dados, mas somente com o documento, em formato JSON, onde estão armazenados os dados de produto.

Problemas detectados no banco de dados corrompido

1. Nomes

Todos os nomes de produto tiveram alguns caracteres modificados, houve substituição de todos os "a" por "æ", "c" por "ç", "o" por "ø", "b" por "ß". É preciso reverter essas substituições para recuperar os nomes originais.

Exemplo:

Original:

```
"name": "iPhone XS Max Prata, com Tela de 6,5, 4G, 64 GB e Câmera de 12 MP"
```

Corrompido:

```
"name": "iPhøne XS Mæx Prætæ, cøm Telæ de 6,5, 4G, 64 GB e Câmeræ de 12 MP"
```

1. Preços

Os preços dos produtos devem ser sempre do tipo number, mas alguns deles estão no tipo string. É necessário transformar as strings novamente em number.

Exemplo:

Original:

"price": 1250.00

Corrompido:

"price": "1250.00"

1. Quantidades

Nos produtos onde a quantidade em estoque era zero, o atributo "quantity" sumiu. Ele precisa existir em todos os produtos, mesmo naqueles em que o estoque é 0.

Exemplo:

Original:

"name": "Conjunto de Panelas Antiaderentes com 05 Peças Paris",
"quantity": 0,
"price": 192.84

Corrompido:

"name": "Conjunto de Panelas Antiaderentes com 05 Peças Paris",
"price": 192.84

A Solução

Funções

As funções desenvolvidas para resolver este problema foram as seguintes:

Primeira função:

```
function normalizarNome(nome) {
    var dicionario_caracteres = {
        "æ": "a",
        "ß": "b",
        "¢": "c",
        "ø": "o"
    }

    return nome.replace(/æ|ß|¢|ø/g, function(char_invalido) {
        return dicionario_caracteres[char_invalido];
    });
```

A função *normalizarNome* vai realizar as seguintes operações

- A variável dicionario_caracteres possui a relação entre os caracteres inválidos e válidos respectivamente;
- A busca pelos caracteres inválidos é realizada através da função replace do JavaScript. Com a expressão regular (regex) utilizada, para cada caracter encontrado buscamos o caracter válido na variável dicionario_caracter.;

Sendo assim, esta função resolve o primeiro problema que são os nomes corrompidos com caracteres especiais inválidos.

Segunda Função:

```
function normalizarPreco(preco) {
    return parseFloat(preco);
}
```

A função *normalizarPreco* é responsável por converter o preço de string para numérico . A conversão é realizada pela função parseFloat do JavaScript que transformará os preços que estão em String em numéricos e assim resolvendo o segundo problema que era dos atributos “price” estar como String.

Terceira Função:

```
function normalizarQuantidade(quantidade) {
    if (quantidade == null) {
        return 0;
    } else {
        return quantidade;
    }
}
```

A função *normalizarQuantidade* é responsável por corrigir a inconsistência com o atributo “quantity” que foi apagado de alguns produtos. Se a quantidade for nulo, ela irá retornar o valor zero, se não a quantidade existente. E assim o terceiro problema é resolvido..

Quarta função:

```
function normalizarBancoDeDados(broken) {
    for (let i = 0; i < broken.length; i++) {
        broken[i].name = normalizarNome(broken[i].name);
        broken[i].price = normalizarPreco(broken[i].price);
        broken[i].quantity = normalizarQuantidade(broken[i].quantity)

    }
}
```

E na função *normalizarBancoDeDados* é onde as funções anteriores são utilizadas para realizar a correção do banco de dados. Esta função recebe o banco de dados corrompido, e por meio do laço de repetição for, é aplicada a normalização de cada produto. Então essa função é responsável por aplicar todas as correções ao banco de dados corrompido.

Quinta função:

```
const fs = require('fs');

/**Lê o arquivo e normaliza o json */
fs.readFile('broken-database.json', 'utf-8', (error, data) => {
    var broken = JSON.parse(data);

    normalizarBancoDeDados(broken);
    console.log(broken);
    ordenarLista(broken);
    console.log(broken);
    calcularValorTotalEmEstoque(broken);

})
```

Esta função é responsável pela leitura do arquivo em JSON. Para fazer isto foi necessário usar o “fs”, uma biblioteca do Node que auxilia na leitura de arquivos..

A função lê o arquivo JSON e dentro da função o arquivo é transformado em um objeto JavaScript. Depois a função normalizarBancoDeDados é invocada, corrigindo assim o banco de dados e por último calculamos o total do estoque através da função calcularValorTotalEmEstoque.

Validação do Banco De Dados:

```
function ordenarLista(broken) {
    let resultado = broken.sort(function(broken1, broken2) {
        let resultado2 = broken1["category"].localeCompare(broken2["category"]);
        if (resultado2 == 0) {
            return broken1["id"] - broken2["id"];
        }
        return resultado2;
    })
}
```

A responsabilidade da função ordenarLista é primeiramente ordenar o banco de dados corrigido por categoria em ordem alfabética e depois por id, ambos em ordem crescente..

A lista é ordenada utilizando a função sort() do JavaScript. A função utilizada pelo sort compara as categorias por nome utilizando o localCompare, porém caso elas

sejam iguais também é realizada a comparação por id. Com isso a função retorna o resultado da ordenação final pelos dois fatores.

```
function calcularValorTotalEmEstoque(broken) {

    let totalPorCategoria = broken.reduce((totalPorCategoria, produto) => {
        let categoriaProdutoAtual = produto['category']
        let totalCategoriaAtual = totalPorCategoria[categoriaProdutoAtual]
        let totalProduto = produto["price"] * produto["quantity"]

        if (totalCategoriaAtual) {
            totalPorCategoria[categoriaProdutoAtual] = totalPorCategoria[categoriaProdutoAtual] + totalProduto;

        } else {
            totalPorCategoria[categoriaProdutoAtual] = totalProduto
        }
        return totalPorCategoria
    }, [])

    for (var categoria in totalPorCategoria) {
        console.log("O total da categoria " + categoria + " é " + totalPorCategoria[categoria])
    }
}
```

Nesta segunda função o propósito era somar o valor total em estoque por categoria, ou seja, percorrer os produtos multiplicando o preço pela quantidade, obtendo assim o valor total dos produtos em estoque.

O valor total por categoria foi calculado utilizando o reduce do JavaScript onde o resultado final esperado é um mapa como abaixo:

{
  "Acessórios": 10,
  "Eletrônicos" 20
}

Para cada produto é extraída a categoria na variável categoriaProdutoAtual, caso a categoria já exista no mapa final chamado totalPorCategoria é somado o totalProduto com o valor atual da categoria. Caso contrário, a categoria é adicionada ao mapa com o valor atual do produto.

Tratamentos feitos no código para evitar bugs

Para evitar futuros bugs tomei o cuidado de não colocar nenhum valor fixo na hora de ler os elementos da lista. Porque se caso tivesse feito isso, e no futuro

adicionássemos mais elementos na lista o código ia quebrar, e nesse cenário então o código pode ser usado caso adicione mais elementos.

Escolha da Linguagem

A solução desse problema foi desenvolvida em JavaScript. Escolhi essa linguagem pois é uma linguagem recomendada para desenvolvimento web e como eu estudo desenvolvimento web na parte de FrontEnd e JavaScript, resolver esse problema na linguagem em que eu estudo pareceu ser mais viável.

Links que me ajudaram no projeto:

Manipular JSON: https://youtu.be/w30zWauuoGw

Replace: https://www.devmedia.com.br/javascript-replace-substituindo-valores-em-uma-string/39176

String em int : https://www.horadecodar.com.br/2021/01/21/como-converter-uma-string-para-int-em-javascript/

Reduce: https://raullesteves.medium.com/javascript-entendendo-o-reduce-de-uma-vez-por-todas-c4cbaa16e380

Sort: https://programadriano.medium.com/javascript-sort-5154c8722e44