Olá, eu sou o Igor e irei apresentar como fiz a estrutura do meu código;

Esse é o corpo principal do código:

```
const data = require('./broken-database.json');
let newData = []
let id = 0
let tempName = []
let name = ''
let quantity = 0
let price = 0
let category = ''

createNewDatabase(data,id,tempName,name,quantity,price,category,newData)

//https://pt.stackoverflow.com/questions/342502/como-criar-um-arquivo-json-a-partir-de-um-js
let fs = require('fs');
fs.writeFileSync('saída.json', JSON.stringify(newData,null,'\t'));

const validate = require('./saída.json');
validação1(validate)
validação2(validate)
```

Primeiramente iniciei uma nova constante ‘data’ que receberá os valores do banco de dados interno baixados no computador, pois foi pedido para não desenvolver qualquer página .html, impossibilitando o uso do jQuery e logo o uso das funções que permite receber valores diretamente do site, como getJSON, se houver uma outra solução sem usar funções jQuery ficaria feliz de receber um retorno com elas.

Também iniciei um novo array que será usado para receber o novo banco de dados já corrigido e as variáveis que temporariamente alocarão os valores que serão passados para o novo banco de dados, a onde cada uma será melhor explicada na próxima seção.

Após isso é feito a chamada da função ‘createNewDatabase’, que receberá usando as variáveis anteriormente criadas, retornará para o array ‘newData’ os banco de dados corrigido.

Será então usado o método usando ‘fs’ para salvar o arquivo ‘saída.json’, que usei o código apresentado em https://pt.stackoverflow.com/questions/342502/como-criar-um-arquivo-json-a-partir-de-um-js para salva-lo, a onde se utiliza das funções da biblioteca JSON para criar um arquivo Sincronizado (pois iremos utilizar ele logo em seguida) usando a função .writeFileSync, pois ela também funciona como overwrite para caso de rodar varias vezes o programa sem ficar colocando entradas repetidas no json, com JSON.stringify transformando o array ‘newData’ com todos os seus valores em um arquivo .json, com a função de transformação nula pois não usei nenhuma transformação e um ‘\t’ para tabular os valores.

Logo em seguida é feito as entrada para as duas validações, a onde a constante ‘validate’ recebe os valores do arquivo ‘saída.json’ e a constante validate será passada pelas duas funções de validação, ‘Validação1’ e ‘Validação2’

Função criarNewDatabase:

```
function createNewDatabase(data,id,tempName,name,quantity,price,category,newData){
    for(let i = 0;i<data.length;i++)
        {
            id = data[i].id;
            tempName = data[i].name.split("");
            name = '';
            name = reparoNomes (name,tempName)

            if(data[i].quantity == undefined)
                quantity = 0
                else
                quantity = data[i].quantity;

            price = parseFloat(data[i].price);

            category = data[i].category;

            let newEntry = {
                "id" : id,
                "name" : name,
                "quantity" : quantity,
                "price": price,
                "category": category,
                }
            newData.push(newEntry)
        }
}
```

Essa função recebe os valores de data, id, tempName, name, quantity, price, category e newData, nele criei um loop de ‘for’ com o limitador da variável de loop ‘i’ sendo o tamanho do banco de dados ‘data’, com cada valor data[i].propriedade é passada para uma variável temporária sem mudança no caso das propriedades .id e .category, com os casos específicos sendo:

- .price: todos os valores são passados para float antes de serem alocados na variável ‘price’, arrumando o problema de varíaveis serem strings
- .quantity: entra numa função lógica ‘if’, com a definição de ‘data[i].quantity == undefined’, se ela for verdadeira, a variável ‘quantity’ = 0, se não ela recebe o valor de data[i].quantity
- .name: primeiro será criado uma array de string, quebrando os valores de data[i].name usando a função split e passando cada letra para uma posição do array tempName, e zerando o valor da variável ‘name’ para não ocorrer erros nas próximas iterações do loop for, após isso será passado o valor de retorno da função reparoNomes para a variável ‘name’

No final será criado um novo objeto newEntry com cada propriedade recebendo os valroes de sua respectiva variável, que será então alocado para dentro do array newData, criando assim um novo banco de dados

Função reparoNomes:

```
function reparoNomes (name,tempName)
    {
        for (let j = 0; j <tempName.length;j++)
        {
            switch (tempName[j] )
            {
                case 'æ':
                    tempName[j] = 'a';
                    name += tempName[j] ;
                    break;

                case '¢':
                    tempName[j]  = 'c';
                    name += tempName[j]
                    break;

                case 'ø':
                    tempName[j]  = 'o';
                    name += tempName[j]
                    break;

                 case 'ß':
                    tempName[j]  = 'b';
                    name += tempName[j]
                    break;

                default:
                    name += tempName[j]
                    break;
            }
        }
        return name;
}
```

Essa função é a mais específica e recebe os valores de name e tempName, primeiro será criado um novo loop ‘for’ , com o valor de parada sendo o tamanho do array tempName, dentro dele será feito um switch:case para cada valor do array tempName, um para cada casa de erro, a onde será substituído " æ " por "a", "¢" por "c", " ø " por "o", " ß " por "b" e se não houver erros passará o valor original, sendo todos concatenado ao valor de name, tendo como retorno final dessa função o valor de name após o loop terminar

Função de validação 1

```
function validação1 (validate){
    let sortedEstoque= validate.sort((v1,v2)=>{
        let v1f = v1.category.toLowerCase()
        let v2f = v2.category.toLowerCase()
        return v1f <v2f ? -1 :v1f>v2f })
        .sort((v1,v2)=>{
            if (v1.category == v2.category){
                return v1.id<v2.id ? -1:v1.id>v2.id
            }
        })
    console.log('\nValidação 1: Banco de dados em ordem de categoria e id\n',sortedEstoque.map(v1=>{
        return {name: v1.name, category: v1.category, id: v1.id, price: v1.price, quantity: v1.quantity}
    }))
}
```

Para fazer essa função me utilizei dos comandos tirados da documentação:

https://www.w3schools.com/jsref/jsref_obj_array.asp

Essa função recebe o valor da constante validate que contém o banco de dados já corrigido.

Primeiro será iniciado uma nova variável ‘sortedEstoqueCat’ que recebe o valor do banco de dados validate após passar pela função .sort, que dentro dela é feita uma arrow function iniciando com dois valores, v1 e v2, a onde será passado para as varíaveis v1f e v2f os valores de v1.category e v2.category respectivamente, tendo a função .toLowerCase() para evitar erros de comparação, a onde ela terá o retorno de verdadeiro se v1f < v2f, ou seja, v1f for antes de v2f em ordem alfabética e falso na situação contrária, a onde será feito o .sort até essa propriedade ser sempre verdadeira.

Após ter feito o primeiro .sort será feito um segundo .sort que só terá mudança na ordem se v1.category.toLowerCase() é igual a v2.category.toLowerCase(), com o toLowerCase() para evitar erros na comparação, a onde terá um retorno verdadeiro se v1.id < v2.id ou seja, v1f for antes de v2f em ordem numérica e falso na situação contrária.

No final aparecerá no terminal qual validação é e o banco de dados mapeado de forma a ficar mais fácil de ver as propriedades .nome, .category e .id para melhor leitura.

Função de Validação 2:

```
function validação2 (validate)
{
    let k = 0;
    let count = 0
    let catQuantity = []
    let countCategory = validate.sort((v1,v2)=>{
                let v1f = v1.category.toLowerCase()
                let v2f = v2.category.toLowerCase()
                return v1f <v2f ? -1 :v1f>v2f
        }).map(v1 =>{
                return {category: v1.category, quantity: v1.quantity}
        })
    for(let i = 0; i<(countCategory.length);i++){
        let j = i + 1
        if(i == (countCategory.length - 1)){
            j = i - 1
        }
        let c1 = countCategory[i].category.toLowerCase()
        let c2 = countCategory[j].category.toLowerCase()
        count += countCategory[i].quantity
        if (c1 != c2){
            catQuantity[k] = {category: countCategory[i].category, quantity: count}
                count = 0
                k++
        }
    }
    console.log('\nValidação 2: Quantidade de produtos por categoria\n', catQuantity)
}
```

Sendo bem sincero não acho que essa função está bem otimizado mas é a melhor maneira que encontrei de monta-lo, se possível gostaria de saber um modo mais compacto e otimizado de escreve-lo

Primeiro iniciei as variáveis k, count e catQuantity, depois iniciei outra variável ‘countCategory’ que recebe o valor de validade após passar por uma função .sort para deixar as categorias em ordem alfabética e uma função .map para retornar somente as categorias e quantidades

Em seguida é feito um loop ‘for’ com o valor de parado sendo o tamanho de ‘countCategory’, iniciando uma nova variável ‘j’ dentro que será uma posição a frente de i e no caso de ir sendo igual a posição final de countCategory, ‘j’ será uma menor para poder fazer a comparação

Após isso será iniciada uma nova variável c1 e c2 que receberá o valor de countCategory[i].category.toLowerCase() e countCategory[j].category.toLowerCase(), respectivamente, tendo o toLowerCase() por motivos já explicados, e count será somado e ao valor anterior de count + countCategory[i].quantity.

Para criar a diferença de categorias e não somente contar todas, fiz uma função logica ‘if’ a onde se c1 for diferente de c2, catQuantity[k] recebe o valor countCategory[i].category, zera o valor de count e aumenta em um o valor de k, fazendo com que passe para o próximo valor da array

No final será mostrado no terminal qual a função e o array catQuantity mostrando todas as categorias e a quantidade de produtos no estoque por categoria