Breve explicação do código

O código que foi enviado é composto por 11 funções, sendo elas divididas em funções: Primordiais, Parte 1 e Parte 2

Primordiais

As funções primordiais são as funções que efetuam as operações de leitura, escrita e atualização do arquivo JSON do banco de dados e elas são:

· WriteJSON
  Escreve no arquivo JSON.
· ReadJSON
  Lê o arquivo JSON.
· UpdateJSON
  Atualiza o arquivo JSON.

Parte 1

As funções da parte 1 são as funções que realizam a primeira parte do teste que vai desde a correção dos bugs no arquivo a exportação do arquivo saída.json, sendo elas:

· changeName
  Responsável pela correção das letras trocadas nos nomes dos produtos
·
changetPrice.
  Responsável pela correção dos preços que não estão no tipo numérico no banco

· changeQuantity
  Responsavel pela correção dos elementos que não possuem o atributo “quantity”

· exportJSON
  Responsável pela exportação do arquivo saida.json

Parte 2

As funções da parte 2 são as funções que realizam a segunda parte do teste que são as validações do arquivo do banco.

· Verify

Acaba sendo “Flag” para saber se o exportJSON já foi realizado e assim possibilitando se as funções dependentes dela possam ser executadas

· categoryQuantity

Reponsável por exibir no console o cálculo da quantidade de cada produto por categoria.

· productList

Reponsável por exibir no console o nome do produto ordenado por ID crescente e Categoria em ordem alfabética.

· allVerify

Reponsável por executar as funções categoryQuantity e productList

Explicação da execução do Teste

PARTE 1

Para a execução do código, inicialmente foram criadas 3 funções que são utilizadas na manipulação do JSON que são: ReadJSON, WriteJSON, UpdateJSON.

As funções Primordiais em si acabam sendo bem parecidas pois é necessário a utilização de pelo menos dois parâmetros sendo filePath que no código acaba sendo o arquivo broken-database.json e o enconding que utilizamos por padrão o utf-8

ReadJSON

A função ReadJSON necessita das variáveis filePath e enconding, Primeiramente é criada uma promise chamada promiseCallback e dentro dela é utilizada a chamada do módulo fs.readFile, dentro do módulo, que no mesmo são passados os parâmetros filePath, enconding, err e data, dentro da chamada do módulo existe primeiramente um if cara caso haja algum problema inicialmente a promise seja rejeita e o código se encerre ali, após isso é utilizado um try catch para que possíveis problemas possam ser detectados e assim rejeitando a promise, quando entramos dentro do try temos o parâmetro dados que pega os valores do JSON a partir do método JSON.parse e assim passando os dados para o parâmetro object que após isso é retornado com o resolve e logo após criando uma nova promise e retornado o valor do object.

WriteJSON

A função writeJSON tem um princípio similar ao da ReadJSON, só alterando o fato de ter o atributo data como adição. A função recebe os parâmetros e dentro dela se utiliza do módulo fs.writeFile que acaba recebendo uma string que será o conteúdo a ser inserido no arquivo JSON, com isso é utilizado o Stringify para converter o conteúdo passado para uma string JSON e assim retornado a resolve com um true.

UpdateJSON

A função UpdateJSON acaba por juntar as funções acima citadas, mas com a adição de um novo atributo sendo o newData, que são as informações a serem atualizadas dentro do arquivo JSON, no início da função fazemos a chamada da função ReadJSON onde fazemos uma promisseCallback se utilizando do async que nos possibilida utiliza o await na ReadJSON que passa a enviar os valores para a const data para ter o objeto que será modificado a seguir é criada a const result onde pega as informações da newData e a envia para a data e após isso fazemos a chamada da função WriteJSON que escrevera as atualizações das informações do arquivo

As funções da parte em si também acabam sendo bem parecidas todas elas contendo os parâmetro filePath que é o caminho do arquivo JSON do banco e o objectJSON, além disso todas elas acabam se aproveitando do método map que acaba por nos ajudar na manipulação para efetuarmos a correção dos bugs.

changeName

A changeName se utiliza do método replace e de regex para poder efetuar a troca dos caracteres.

changePrice

A changePrice se utiliza do método parseFloat para trocar os atributos preços que nao estao em numérico para numérico sendo especificamente o Float por se utilizar de números “quebrados”

changeQuantity

Para a changeQuantity a lógica acabamos por verificar se o objeto possui o atributo quantity, para isso com o map foi utilizado a hasOwnProperty junto com a chave do objeto para ser possível a verificação se existe o atributo e quando não era atribuído o valor 0

exportJSON

A exportJSON tem como parâmetros o caminho do arquivo JSON e o caminho onde será salvo o arquivo saida.json e a função acaba se utilizando da async para assim se utilizar do await na chamada do ReadJSON, após isso ocorre a chamada das funções com os paramentos do caminho dao arquivo JSON e o objeto criado pelo ReadJSON para fazer as correções necessárias no arquivo JSON, após isso usamos um try catch, onde utilizamos a função WriteJSON para a criação do arquivo saida com base no arquivo anterior já corrigido.

PARTE 2

Após a correção dos bugs e criação do arquivo saída, agora é necessário fazer as validações no arquivo saída, para sabermos se foi efetivo as correções dos bugs, mas para isso é essencial que as funções de validação sejam executadas após o termino da função ExportJSON.

verify

A função verify acaba sendo responsável pela detecção se a ExportJSON já terminou sua execução ou não, para isso criamos a const status onde é utilizado o async para podermos se utiliza do await na chamada da ExportJSON e pegamos o retorno da função que pode ser variado em true e false

categoryQuantity

Nessa função fazemos a somatória de produtos, para isso é necessário criar variáveis para cada categoria de produto, após isso se utilizam do map para poder manipular o objeto e como isso se utilizamos de alguns ifs para auxiliar a divisão entre as categorias e por fim imprimimos no console as variáveis.

productList

Primeiramente se utilizamos da const objectJSON onde se utilizam do async para fazer a chamada da função ReadJSON, após isso utilizamos a const listAll onde se utilizamos do método map para poder manipular o objeto após isso retornaremos os atributos category, id e name como uma string e depois imprimimos a const listAll no console com o método sort para ordenar

allVerify

Primeiramente na const allVerify se utilizamos do async para poder se utilizar do await na chamada da função verify, após isso temos um if onde verificamos se o retorno foi true, caso seja fazemos a chamadas das funções caregoryQuantity e productList

Apresentação e Considerações

Primeiramente agradeço a nova oportunidade de envio do teste apesar do tempo passado e também pela oportunidade em si, pois foi um bom desafio para testar meus conhecimento e aprender mais tanto sobre javascript e lógica.

Sobre mim meu nome è Gianluca, tenho 23 anos, estudo engenharia de computação na FACENS no 6º Semestre que vai se iniciar em agosto, atualmente eu trabalho com desenvolvimento de robotização com python mas tenho como objetivo uma “mudança” de área que possa me ajudar nos objetivos de carreira, gostaria muito de ter a oportunidade de fazer uma entrevista pois acredito que na rocky posso ter uma evolução tanto profissional quanto pessoal.

Referências

https://www.youtube.com/watch?v=w30zWauuoGw&t=1664s
https://www.freecodecamp.org/portuguese/news/map-em-javascript-como-usar-a-funcao-map-do-js-metodo-de-arrays/#:~:text=A%20sintaxe%20completa%20do%20m%C3%A9todo,para%20cada%20elemento%20do%20array.

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/HasOwnProperty

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/JSON

https://www.rexegg.com/regex-quickstart.html

https://www.edsonemiliano.com.br/blog/como-ordenar-uma-array-de objetos-com-javascript-sort