## Documentação

O projeto tem como base a normalização de arquivos JSON e após isto uma normalização a partir dos dados obtidos em código, vale ressaltar que para o desenvolvimento do código foi utilizado somente JS e as bibliotecas nativas do node.js, logo não foi utilizado no projeto nenhuma biblioteca terceira.

## 1 - Estrutura do projeto

O projeto foi dividido e estruturado em subpastas que facilitam a manutenibilidade do código, como pode ser visto na imagem abaixo:



#### Pasta common

Responsável por guardas as variáveis que poderiam ser utilizadas em demais pontos do nosso código.

#### Pasta csv

Pasta com o CSV gerado para a análise de dados posteriores

#### Pasta data

Contém os arquivos JSON corrompidos e os arquivos normalizados da aplicação, foi também incluído uma cópia do CSV em formato JSON.

#### Pasta SQL

Contém os SQL’s utilizados no projeto, sendo eles a inserção e criação das tabelas, e a unificação dos JSON gerados em uma tabela unificada.

#### Pasta Utils

Responsável por conter as funções que podem ser utilizadas em diversos pontos do código, evitando assim reescrever novamente.

#### Arquivo index

Este é o arquivo executado ao iniciar a aplicação e conta com a lógica para a chamada de demais funções necessárias para a normalização do arquivo.

## 2 - Lógica do projeto

### Função main

A função no arquivo `main.js` é responsável por chamar todas as outras funções e variáveis utilizadas no código, nela temos dois cenários:

* Normalização com sucesso:
  + Chama uma função responsável pela leitura dos dois arquivos JSON’s e salva estes dados em uma constante
  + Cria logs de início da aplicação
  + Chama a função que faz a normalização dos dados e salva os dados em uma constante.
  + Entra em um bloco de try/catch para evitar erros ao salvar os arquivos e conseguir tratar os erros que acontecem.
  + Chama outra função responsável por salvar os arquivos normalizados no projeto
  + Logs de sucesso ou falha ao salvar os arquivos.

### Arquivo utils/data

Pasta responsável por funções que genéricas ao ponto de poderem ser reaproveitadas em demais partes do código, facilitando assim manutenibilidade, escalabilidade e legibilidade.

#### Função readFiles

Primeira função chamada no arquivo main, nela temos um bloco try/catch para tratamento de erros e utilizamos a biblioteca do node fs (file system) e sua função readFile, responsável por ler um arquivo do sistema.

Nela também temos um parâmetro que representa o nome do arquivo a ser lido no sistema e usamos o path.resolve(\_\_dirname, “nome\_arquivo”) ao manipular caminhos de arquivos no Node.js para evitar erros relacionados a caminhos relativos, garantindo que o caminho seja resolvido de forma correta, independente do diretório de execução, prevenindo inconsistências e problemas de localização de arquivos.

#### Função normalizeData

A função principal do projeto responsável pela lógica da normalização dos dados, recebemos um parâmetro chamado “data” que representa o arquivo lido pela função “readFile”, sua lógica é composta por:

1. Bloco try/catch para tratamento dos erros
2. Uma constante chamada “dataToNormalize”, que converte o arquivo JSON em um array de objetos para serem manipulados na função.
3. Constante normalizedData responsável pela manipulação dos dados, nela fazemos um .map da constante “dataToNormalize”, e pegamos cada um dos itens do JSON para serem manipulados, e em seguida:
   1. fazemos uma desestruturação em uma constante “normalizedItem” que representa um item do objeto a ser normalizado.
   2. Em seguida, temos uma constante chamada "KeysToNormalize", que é exportada de outro arquivo. Nós a percorremos usando um loop forEach e verificamos se o nome da propriedade no JSON está presente nessa constante. Se for verdadeiro, substituímos a letra incorreta pela correta usando a função "replace" nativa do JavaScript. Para isso, utilizamos uma constante chamada "NormalizeDataMap" que mapeia quais letras devem ser substituídas. É importante mencionar que criamos uma expressão regular (regex) que lê as ocorrências globalmente, garantindo que, se houver mais de uma letra incorreta em uma palavra, a regex substituirá todas as ocorrências.
   3. Após garantirmos que todas as palavras necessárias foram corrigidas, o próximo passo é normalizar a propriedade "vendas", onde alguns itens foram corrompidos e se transformaram em strings. Para lidar com isso, criamos uma constante chamada "KeysToNumber" que lista as colunas em que os valores devem ser exclusivamente numéricos. Em seguida, percorremos essa lista usando um loop forEach e verificamos se estamos na propriedade do objeto em que precisamos converter para número. Se for verdadeiro, utilizamos a função Number() do JavaScript para realizar essa conversão.
   4. Retornamos o valor de “normalizedItem” que representa o item do objeto já totalmente normalizado para a constante “normalizedData”
4. Retornamos o valor de “normalizedData” onde temos todos os valores normalizados e prontos para analise de dados.
5. Informamos ao usuário que a operação foi bem-sucedida se tudo acontecer como esperado e, se houver algum problema, fornecemos uma mensagem de erro amigável para facilitar o entendimento.

#### Função saveData

A função saveData recebe dados (data) e um nome de arquivo (fileName). Ela tenta escrever esses dados em um arquivo JSON no sistema de arquivos. Primeiro, verifica se os dados existem; caso contrário, lança um erro indicando que o arquivo não foi salvo. Em seguida, utiliza a função fs.writeFile para escrever os dados no arquivo correspondente no diretório "../data". Se a operação for bem-sucedida, imprime uma mensagem de sucesso no console. Se ocorrer um erro durante o processo, captura esse erro, imprime uma mensagem de erro detalhada no console e relança o erro para ser tratado por quem chamar a função. Essa função serve para salvar dados em arquivos JSON de forma assíncrona, lidando com possíveis erros durante o processo.

### Arquivo common/normalize

Responsável por exportar 3 constantes:

1. KeysToNormalize: representa quais propriedades do JSON tiveram precisam ser normalizados, com ela sabemos quais propriedades devemos aplicar a lógica de troca de letras, para que a string possa ser compreendida, ela serve como um guia para aplicar a lógica de correção apenas nas propriedades relevantes, tornando o processo mais eficiente e direcionado.
2. KeysToNumber: Possue a mesma ideia da variável acima, entretanto representa quais propriedades do JSON devem sem exclusivamente números.
3. NormalizeDataMap: é uma lista (array) de objetos que serve para mapear quais são os valores incorretos e indicar quais devem ser os valores corretos durante o processo de normalização. Cada objeto nesta lista possui duas propriedades:
   1. from: que mapeia a string que está corrompida/incorreta, e
   2. to: que representa o valor correto que deveria estar na string.

Assim, essa estrutura nos ajuda a associar os valores que precisam ser corrigidos aos valores corretos, facilitando a lógica de normalização das strings.

Com isso finalizamos a parte de normalização de dados e estamos prontos para a segunda parte do teste.

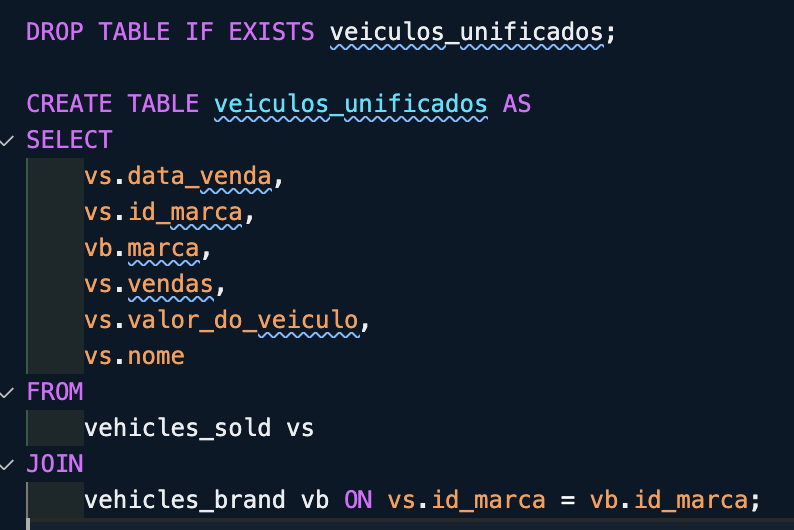
## SQL

Foi solicitado após a normalização que os dados fossem importados no site [SQL Online](https://sqliteonline.com/), foi feita a importação dos dois arquivos gerados anteriormente na plataforma, mudando os nomes das colunas de cada tabela para o que estava representado no JSON, evitando assim os nomes genéricos sugeridos pela plataforma tais como c1, c2, c3.

Além disso, incluímos uma regra SQL "DROP TABLE IF EXISTS" para prevenir erros durante possíveis migrações no banco de dados. Essa medida assegura que, caso a tabela já exista, ela será excluída, evitando possíveis problemas na aplicação. Isso ajuda a garantir uma transição suave e consistente, prevenindo conflitos que poderiam ocorrer se a tabela já estivesse presente durante a execução das migrações

### Unificação das tabelas

Foi solicitado no teste a junção das duas tabelas geradas em uma tabela única, para isso foi feito o código abaixo:



Assim criamos uma nova tabela chamada "veiculos\_unificados". Os dados desta nova tabela são provenientes da junção (JOIN) das tabelas "vehicles\_sold" (representada por "vs") e "vehicles\_brand" (representada por "vb"). A junção é feita usando a condição de igualdade entre as colunas "id\_marca" de ambas as tabelas. A tabela resultante ("veiculos\_unificados") contém as colunas selecionadas da tabela "vehicles\_sold", como "data\_venda", "id\_marca", "vendas", "valor\_do\_veiculo" e "nome", juntamente com a coluna "marca" da tabela "vehicles\_brand". Ou seja, ela unifica informações relevantes sobre veículos vendidos, incluindo dados sobre a marca associada a cada venda.

Com isso já temos todos os dados em uma única tabela e estamos prontos para a última parte do teste, que será a análise de dados, para isso exportamos a tabela unificada como um arquivo de extensão “CSV” na propiá ferramenta [SQL Online](https://sqliteonline.com/).

## Relatório dos dados

Para fazer o relatório, foi utilizado o Google Sheets visto que é uma ferramenta simples e fácil de usar, o que a torna eficaz na criação de gráficos e análise de dados. Essa escolha foi feita para tornar a compreensão dos dados e a elaboração de relatórios mais acessíveis e intuitivas. O Google Sheets proporciona uma experiência amigável, facilitando o processo de organização e visualização das informações para a criação de relatórios de forma descomplicada.

No processo, importamos o arquivo gerado no SQL Online para criar uma tabela consolidada e normalizada com todos os dados. Em seguida, criamos gráficos para cada uma das perguntas do desafio. É importante destacar que, para responder à quarta pergunta (Quais marcas geraram uma receita maior com número menor de vendas?), foi gerada uma nova coluna chamada "receita". Essa coluna é calculada com base no valor do veículo e no número de vendas. A inclusão dessa coluna facilita a resposta à quarta pergunta, permitindo uma análise mais eficiente sobre quais marcas obtiveram uma receita superior mesmo com um número menor de vendas. Essa abordagem visa tornar a compreensão e interpretação dos dados mais acessíveis no contexto do desafio proposto.