aiko

# Documentação Estapa de Teste Processo Seletivo

# Arthur Gomides Godoy

[Observações: 2](#_Toc11846)

[Views: 3](#_Toc17888)

[-Home 3](#_Toc22992)

[Tratamento de Dados: 4](#_Toc32246)

[-Funções: 5](#_Toc24854)

[Componentes: 7](#_Toc15027)

[-TopBar 7](#_Toc4050)

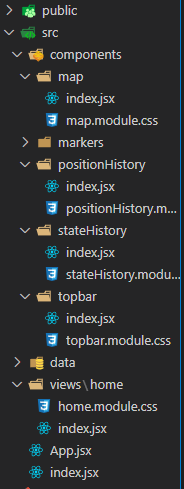
[-Map: 7](#_Toc27075)

[-Markers: 8](#_Toc4726)

[-StateHistory: 9](#_Toc28026)

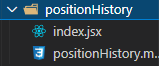
# Observações:

1. O projeto foi feito usando MUI, que já possui vários componentes prontos e estilizados, substituindo as tags HTML padrão, porém acabei usando algumas tags html, para conseguir dar um estilo totalmente diferente.
2. O projeto é organizado em pastas, e cada uma possui seu respectivo index.jsx e caso use CSS teria um arquivo CSS próprio. (vi esse modelo de organização no projeto do EM SUMA, e aderi até então).

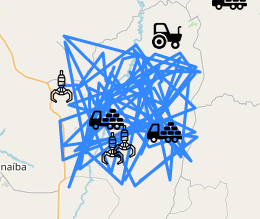


1. Fiz o Histórico de Posições de 3 maneiras:

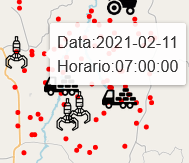
-A primeira seria como uma tabela (igual ao histórico de estados) e mantive essa parte no código, por mais que não esteja utilizando.



-A segunda é mostrando no mapa as posições com uma linha, mas não ficou muito organizado



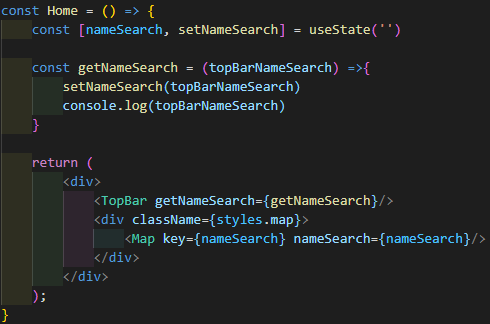
- A terceira são com pontos, das posições anteriores, onde mostra a data da posição. E essa é a mantida no projeto.



# Views:

### -Home

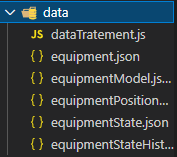
Esse projeto possui somente uma view, que é a Home, o que pode ser que deixa ele um pouco redundante, uma vez que poderia somente carregar os componentes no App.jsx, na Home são renderizados os componentes TopBar e Map.



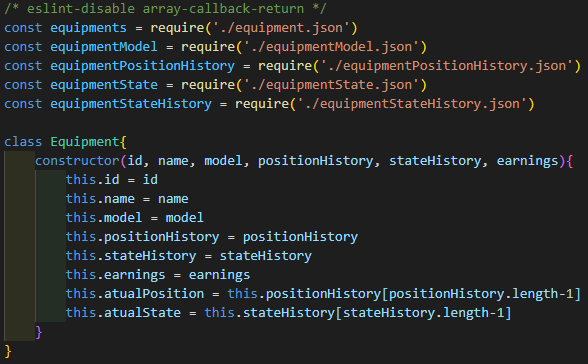
Também possui a função getNameSearch que serve para pegar o valor do Selection do componente TopBar (explicação mais a frente)

# Tratamento de Dados:

Os Dados estão todos guardados na pasta chamada Data. Onde estão todos os arquivos JSON e o arquivo dataTratement.js.



Nesse projeto decidi criar um arquivo separado para organizar todos os meus dados de forma que ficasse muito mais fácil de lidar com eles, não tendo que organizar a cada vez que precisasse dos dados. Por isso criei o dataTratement.js.

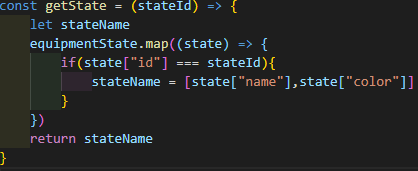


Primeiramente Importei todos os meu arquivos JSON, depois criei uma classe, meu objetivo era ter um objeto do equipamento com todos seus dados prontos e organizados, nele temos o Id, nome, modelo, todo o historico de posição e estado, e também a posição e estado atual, Também capturei todos os dados de ganhos, porém não deu tempo de utiliza-lo.

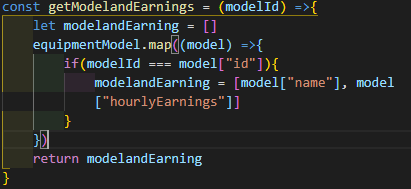
Para conseguir todas as informações precisei criar funçoes separadas para cada propriedade da classe.

### -Funções:

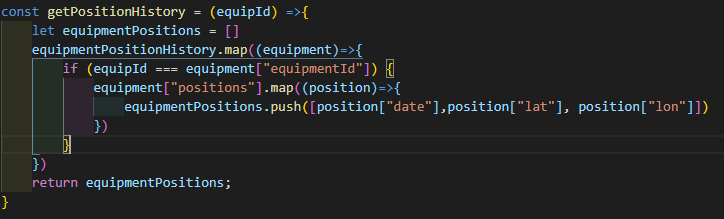
-GetState: A função getState irá receber um idState, e irá retornar um Array, com o nome do estado e a respectiva cor.



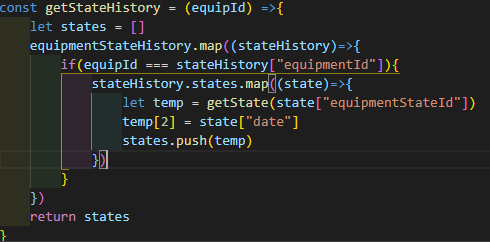
-GetModelandEarnings: Essa função receberá um id de modelo, e irá retornar um array com o nome do modelo e o “hourlyEarning”



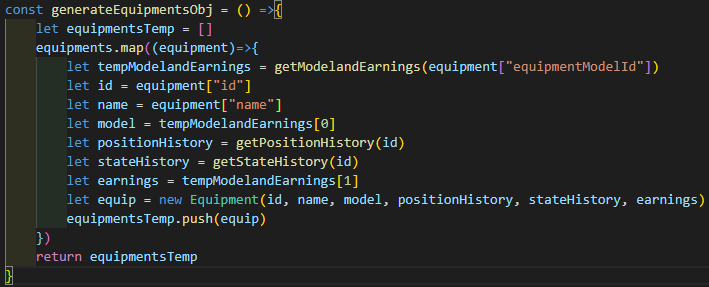
-getPositionHistory: Essa função irá retornar um array com todas as posições do equipamento, cada posição do array principal possui outro array com a data, latitude e lontitude. Essa função precisa de um id de equipamento como parâmetro.



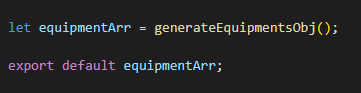
-getStateHistory: Essa função irá retornar um array com todas os estados do equipamento, cada posição do array principal possui outro array com o nome do estado sua cor e data, o nome e cor são pegos da função getState. Essa função precisa de um id de equipamento como parâmetro.



-generateEquipmentsObj: Essa é a função que irá gerar o array de objetos, nele nós temos um .map dos equipments, e depois vai criando variáveis com os valores de retorno das funções anteriores,



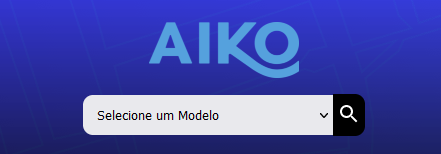
Por fim o dataTreatement irá exportar um Arr de objetos, que é declarado e exportado ao final.



# Componentes:

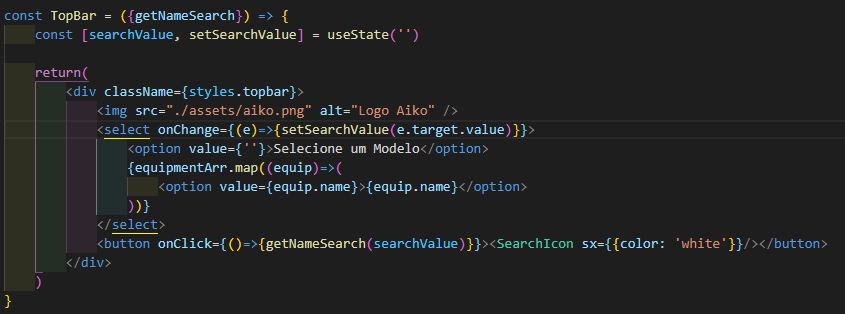
-TopBar

O componente Top Bar, é o componente da barra de navegação de cima.

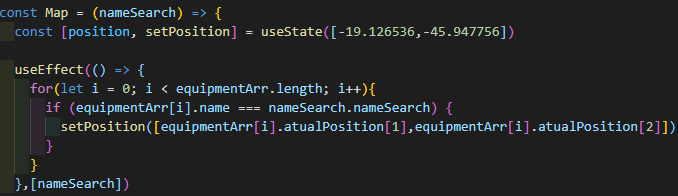


Ele possui um Select onde todos os nomes dos Equipamentos são listados, O nome selecionado é enviado para o componente pai através de uma função.

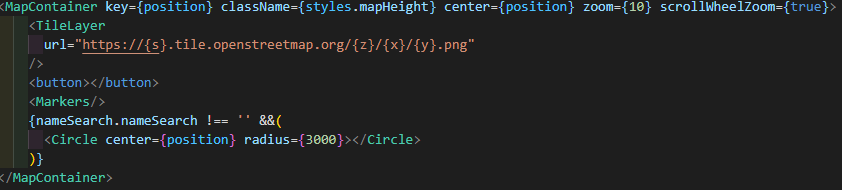
Essa função é passada da view Home para o componente filho TopBar, que por sua vez altera o valor da variável no componente Pai, do componente pai ele é enviado para outro componente filho que é o map, onde ele faz a tratativa para centralizar o respectivo equipamento.



### -Map:

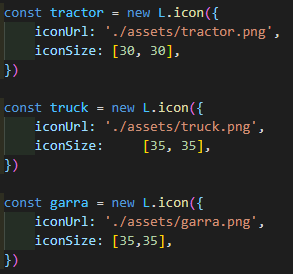
Esse componente irá retornar o mapa, o mapa é carregado da biblioteca do Leaflet, nele nós temos a propriedade nameSearch, que é para mostrar o marcador conrrespondente da pesquisa, para isso tem a variável position com o hook useState, e depois um hook useEffect para pegar a posição através do nome do equipamento. 

Após isso o MapContainer possui uma key igual a position, e será centralizado no position. Caso o nameSeach for diferente de ‘ ‘, ou seja, caso tenha alguma pesquisa, o zoom também será mudado de 10 para 14 e também aparecerá um círculo no marcador.

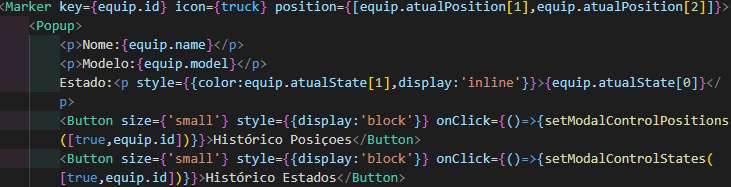


### -Markers:

Esse componente irá retornar os marcadores do mapa, e é o componente com mais funcionalidades na aplicação, primeiramente nós temos três variáveis que são as responsáveis por mudar o tipo de ícone do marcador na aplicação. (Acabei misturando o inglês com o portugues ali kkkk).



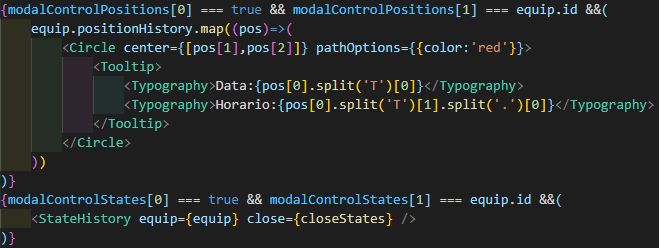
Depois temos um .map do equipmentArr (do DataTratment), esse .map irá passar por 3 if/else, para verificar o tipo do ícone, se irá ser caminhão, garra ou trator, cada um desse retornará um Marker, e um Popup com o nome, modelo e estado, e dois botões, um para ver o historico de posições e outro para o historico de estados.



O historico de posições e de estados tem um if ternário, onde irá renderizar o componente do historico somente se for true, eu estava tendo um problema que eu estava fazendo a mesma verificação para abrir todos, e fechar todos, então quando se abria um ele abria normal, mas depois quando ia abrir outro se abria os 2 juntos, e assim em diante, então resolvi esse problema adicionando mais uma verificação.



Essas variaveis de controle, além de ter o falso (fechado), e true(aberto), vão ter também um Id, esse id será o do Marker dentro do .map. Então agora sempre que eu abro um modal, a variável é alterada para [true, id], e depois fechada para, [false,id], e as verificações ternárias passou a ser se é verdadeiro e se o id da variável é igual ao id do equipamento.



Com certeza existem maneiras melhores, mas essa foi uma gambiarra que encontrei e funcionou muito bem para minha situação. Caso ele clique para ver o historico de posições sé um .map, onde será retornado um circle vermelho, para cada posição do historico.

### -StateHistory:

Esse componente recebe como props uma variável contendo o objeto do equipamento que deverá ser mostrado o historico e também recebe uma função do componente pai (Markers) chamada close, que é usada para fechar o componente.





Ele renderiza um Modal, e dentro desse modal existe uma tabela, e dentro do TableBody, tem um .map onde será renderizado um TableCell, contendo a Data, Hora e Estado. E ao final um botão de fechar que irá usa a função do componente pai, para fecha-lo dentro do próprio componente pai.