Challenge Chart Plot

Esta aplicação tem como objetivo gerar gráficos através da inserção de linhas de código semelhantes a um objeto JSON em um editor de código-fonte ao pressionar um botão.

1. Input: Dentre as diversas bibliotecas disponíveis, optei pelo 'Monaco Editor' que é um editor de código bastante conhecido oferecido pela Microsoft, e sua versão para React pode ser encontrada aqui: https://github.com/suren-atoyan/monaco-react#readme; e uma biblioteca que permite a responsividade do mesmo: https://github.com/gfmio/responsive-react-monaco-editor#readme.

Dado aceito pelo o input: Os dados abaixo têm como base a referência fornecida ao desenvolvedor, sendo eles:

{type: 'start', timestamp: 1519862400000, select: ['min_response_time', 'max_response_time'], group: ['os', 'browser']}

{type: 'span', timestamp: 1519862400000, begin: 1519862400000, end: 1519862460000}

{type: 'data', timestamp: 1519862400000, os: 'linux', browser: 'chrome', min_response_time: 0.1, max_response_time: 1.3}

{type: 'data', timestamp: 1519862400000, os: 'mac', browser: 'chrome', min_response_time: 0.2, max_response_time: 1.2}

{type: 'data', timestamp: 1519862400000, os: 'mac', browser: 'firefox', min_response_time: 0.3, max_response_time: 1.2}

{type: 'data', timestamp: 1519862400000, os: 'linux', browser: 'firefox', min_response_time: 0.1, max_response_time: 1.0}

{type: 'data', timestamp: 1519862460000, os: 'linux', browser: 'chrome', min_response_time: 0.2, max_response_time: 0.9}

{type: 'data', timestamp: 1519862460000, os: 'mac', browser: 'chrome', min_response_time: 0.1, max_response_time: 1.0}

{type: 'data', timestamp: 1519862460000, os: 'mac', browser: 'firefox', min_response_time: 0.2, max_response_time: 1.1}

{type: 'data', timestamp: 1519862460000, os: 'linux', browser: 'firefox', min_response_time: 0.3, max_response_time: 1.4}

{type: 'stop', timestamp: 1519862460000}

Porém quaisquer dados podem ser fornecidos, contanto que sigam o padrão acima. Exemplo:

{type: 'start', timestamp: 1601331894000, select: ['min_response_time', 'max_response_time'], group: ['os', 'browser']}

{type: 'span', timestamp: 1601331894000, begin: 1601331894000, end: 1601331954000}

{type: 'data', timestamp: 1601331894000, os: 'linux', browser: 'chrome', min_response_time: 0.5, max_response_time: 1.5}

{type: 'data', timestamp: 1601331894000, os: 'mac', browser: 'chrome', min_response_time: 0.7, max_response_time: 1.7}

{type: 'data', timestamp: 1601331894000, os: 'mac', browser: 'firefox', min_response_time: 0.6, max_response_time: 1.6}

{type: 'data', timestamp: 1601331894000, os: 'linux', browser: 'firefox', min_response_time: 0.5, max_response_time: 1.5}

{type: 'data', timestamp: 1601331954000, os: 'linux', browser: 'chrome', min_response_time: 0.3, max_response_time: 1.3}

{type: 'data', timestamp: 1601331954000, os: 'mac', browser: 'chrome', min_response_time: 0.4, max_response_time: 1.4}

{type: 'data', timestamp: 1601331954000, os: 'mac', browser: 'firefox', min_response_time: 0.2, max_response_time: 1.2}

{type: 'data', timestamp: 1601331954000, os: 'linux', browser: 'firefox', min_response_time: 0.3, max_response_time: 1.4}

{type: 'stop', timestamp: 1601331954000}

Caso seja inserido um dado que não possa ser convertido para um objeto JSON válido, a aplicação apresentará uma mensagem de erro.

2. Gráfico: A biblioteca Chart.js https://www.chartjs.org/ é simples e relativamente fácil de usar e se adequa com as necessidades da aplicação, ficando como responsável de gerar o gráfico de acordo com

os dados postos no input, mostrando em cada linha os valores de entrada.

3. Botão: O botão tem como finalidade gerar o gráfico da aplicação tendo que reconhecer os dados inseridos no input convertendo-os posteriormente para um Array de objeto JSON.

Fluxo da aplicação:

Após inserir os dados no input e clicar no botão 'Gerenate Chart', os seguintes passos são executados:

1. *generateChart*: função acionada após clicar no botão e tem como objetivo:
   - Transformar os dados recuperados do input em um objeto JSON; **
   - Com os dados já transformados, estes são separados de acordo com seu tipo (start, span, data e stop);
   - Os dados entram em um looping de repetição e toda vez que um tipo "data" é encontrado os valores correspondentes as tags "os", "browser", "min_response_time" ou "max_response_time" são recuperados para montar as linhas que são exibidas no gráfico, com suas respectivas legendas;

** *Fluxo de Exceção*:

- Caso seja inserido um dado inválido, a aplicação retornará uma mensagem de erro ao usuário solicitando a verificação dos dados inseridos por ele;

Exemplo de dado inválido:

{type: 'start', timestamp: 1601331894000, select: ['min_response_time', 'max_response_time'], group: ['os', 'browser']}

{type: 'span', timestamp: 1601331894000, begin: 1601331894000, end: 1601331954000}

{type: 'data', timestamp: 1601331894000, : 'linux', browser: 'chrome', min_response_time: 0.5, max_response_time: 1.5}

{type: 'data', timestamp: 1601331894000, : 'mac', browser: 'chrome', min_response_time: 0.7, max_response_time: 1.7}

{type: 'data', timestamp: 1601331894000, os: 'mac', browser: 'firefox', min_response_time: 0.6, max_response_time: 1.6}

{type: 'data', timestamp: 1601331894000, os: 'linux', browser: 'firefox', min_response_time: 0.5, max_response_time: 1.5}

{type: 'data', timestamp: 1601331954000, os: 'linux', browser: 'chrome', min_response_time: 0.3, max_response_time: 1.3}

{type: 'data', timestamp: 1601331954000, os: 'mac', browser: 'chrome', min_response_time: 0.4, max_response_time: 1.4}

{type: 'data', timestamp: 1601331954000, os: 'mac', browser: 'firefox', min_response_time: 0.2, max_response_time: 1.2}

{type: 'data', timestamp: 1601331954000, os: 'linux', browser: 'firefox', min_response_time: 0.3, max_response_time: 1.4}

{type: 'stop', timestamp: 1601331954000}

Outras bibliotecas:

- *sweetalert2*: Biblioteca usada para criação estilizada e responsiva dos modais de alerta na aplicação.
- *dirty-json:* Biblioteca utilizada para análise e conversão de dados para JSON.

Arquivos:

chartOption.js **=>** Estilização do gráfico.

inputData.js **=>** Recebe os valores do tipo "data" JSON e mapeia os dados transformando-os nas linhas gráfico.

inputParser.js **=>** Recebe o JSON e mapeia os dados transformando-os para os demais elementos do gráfico.

chartView.js **=>** Componente que possui o construtor e métodos de inicialização da biblioteca Chart.js.

dashboardView.js **=>** Componente responsável por agrupar e renderizar todos os elementos da aplicação.

inputView.js **=>** Componente que possui o construtor e métodos de inicialização da biblioteca Monaco Editor.

Importante!

parseStringToJson.test.js **=>** Function criada com o objetivo de realizar o teste unitário na transformação do objeto recebido pelo input em um objeto JSON.

Neste caso cabe ressaltar que por conta da arquitetura escolhida, não foi possível criar mocks dos dados para validação dos testes, uma vez que as cores referentes as linhas da tabela são criadas pelo método Math.random, gerando cores RGB aleatórias.