Challenge Chart Plot

The present application aims at generating charts through code lines insertion similar to a JSON object in a source-code editor by clicking a button.

1. **Input**: Among the several available libraries, I`ve opted for 'Monaco Editor' which is a well-known Microsoft code editor, its React version can be found here: https://github.com/suren-atoyan/monaco-react#readme; And a library which allows its responsiveness: https://github.com/gfmio/responsive-react-monaco-editor#readme.

Data accepted by the input: The following data are based on the reference supplied to the developer, being such:

```
{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}
```

However, any data could be provided as long as they follow the pattern above. Eg:

```
{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}
```

In case of insertion of a datum which could not be converted to a valid JSON object, the application will show an error message.

2. **Chart**: The library Chart.js https://www.chartjs.org/ is simple, relatively easy to use and adaptable to the needs of the application, being responsible for generating the chart according to the data inserted in the input, showing in each line the entry values.

3. **Button**: The button serves to generating the chart of the application recognizing the data inserted in the input converting them later to an Array of JSON object.

Application flow:

After inserting the data in the input and clicking the button “Generate Chart”, the following steps will be executed:

Normal flow:

1. *generateChart*: Function activated after clicking on the button to:
   - Transform the input recuperated data in a JSON object; ******
   - With the data already transformed, they are sorted according to their type (start, span, data and stop);
   - The data start a looping and every time a font “data” is found the tags corresponding values “os”, “browser”, “mini_responde_time” are recuperated to assemble the lines which are displayed on the chart with their corresponding subtitles;

******** *Alternative Flow and exception*:

- In case an invalid datum is inserted, the application will return an error message to the user requiring the verification of the data inserted.

Invalid Data Example:

```
{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}
```

Other Libraries:

- *sweetalert2*: Library used to create stylish and responsive alert modals in the application.
- *dirty-json: Library used to analyse and convert data to JSON.*

Files:

chartOption.js **=>** Chart stylishing.

inputData.js **=>** Gets the values “data" JSON and maps the data transforming them in chart lines.

inputParser.js **=>** Gets JSON and maps the data transforming them for the other elements of the chart.

chartView.js **=>** Component which has the constructor and the methods of initialization of the library Chart.js.

dashboardView.js **=>** Component responsible for the grouping and rendering all the application elements.

inputView.js **=>** Component which has the constructor and method of initializing the library Monaco Editor.

Important!

parseStringToJson.test.js **=>** Function created aiming at performing the unitary test in the transformation of the object received by the input in a JSON object.

In such case, it is worth to mention that due to the chosen architecture it wasn’t possible to make mocks of the data to validate the tests, once the colours referring to the lines of the chart are generated by the Math.random method, generating random RGB colours.