



SportSee Build an analytics dashboard with React

PRESENTATION

SportSee is a startup dedicated to sports coaching. In full growth, the company will today launch a new version of the user profile page. The goal is to redo the profile page with React.

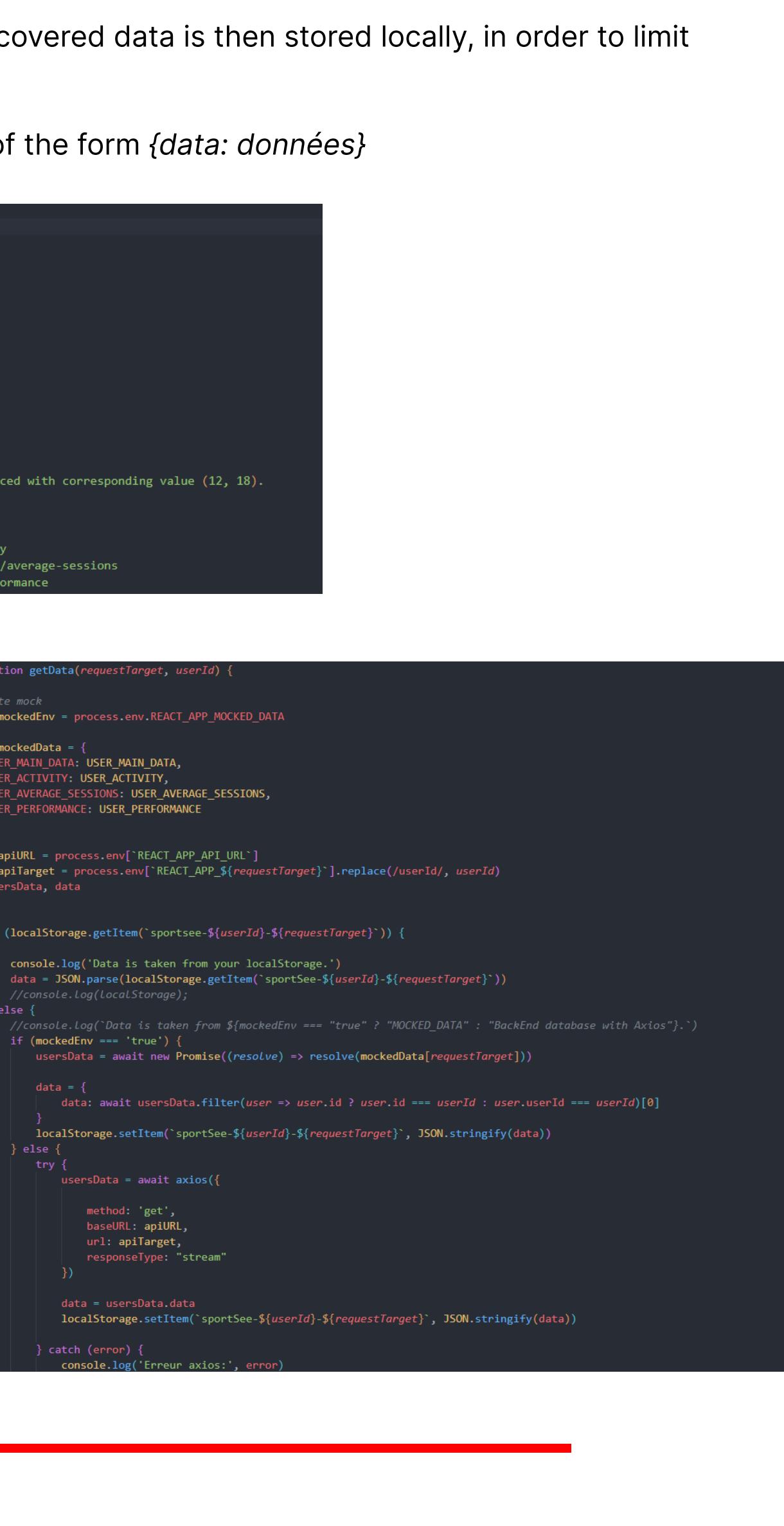
PROJECT

The project is developed as an application, under the React environment, using the **create-react-app** command.
React, React DOM as well as Axios (for data recovery) will be used.
SASS will be used for CSS

only the **Desktop** version was made, according to the following structure:

- **app** : functions
- **assets** : medias
- **components**
- **pages**
- **styles** : scss + css

```
SPORTSEE
> build
> node_modules
> public
> _tests_
> app
> assets
> mockData
  > icon-1.svg
  > icon-2.svg
  > icon-3.svg
  > icon-4.svg
  > logo.png
> components
> dashboard
  < NavHorizontal.jsx
  < NavVertical.jsx
> pages
  < Dashboard.jsx
> styles
  > components
  > pages
  > utils
    # App.css
    # App.css.map
    # App.scss
    # index.css
    # logo.css
    # reportWebVitals.js
    # setupTests.js
  # babelrc
  .env
  .gitignore
  package.json
  README.md
  yarn.lock
```



API REQUESTS

API requests are centralized in the **getData.js** page.
The module is configurable:

- **.env** - this file contains the launch port of the app, the URL containing the BACK-END, the access to the mocked data as well as the request routes for the BACK-END API
- **localStorage** - the recovered data is then stored locally, in order to limit network requests
- **@return** - an object of the form `{data: données}`

```
1 // Define on which port number to start
PORT=4001

// API URL
REACT_APP_API_URL=http://localhost:3000

/* 
 * AXIOS fetching MOCKED DATA
 * @param "true" || "false"
 * @returns The boolean string set up here.
 */
REACT_APP_MOCKED_DATA=false

/*
 * API TARGET (endpoints)
 * @ param userId - the variable to be replaced with corresponding value (12, 18).
 * @ returns The URL part of the API target.
 */
REACT_APP_USER_MAIN_DATA=/user/:userId
REACT_APP_USER_ACTIVITY=/user/:userId/activity
REACT_APP_AVERAGE_SESSIONS=/user/:userId/average-sessions
REACT_APP_USER_PERFORMANCE=/user/:userId/performance
```

```
async function getData(requestTarget, userId) {
  //create mock
  const mockedEnv = process.env.REACT_APP_MOCKED_DATA

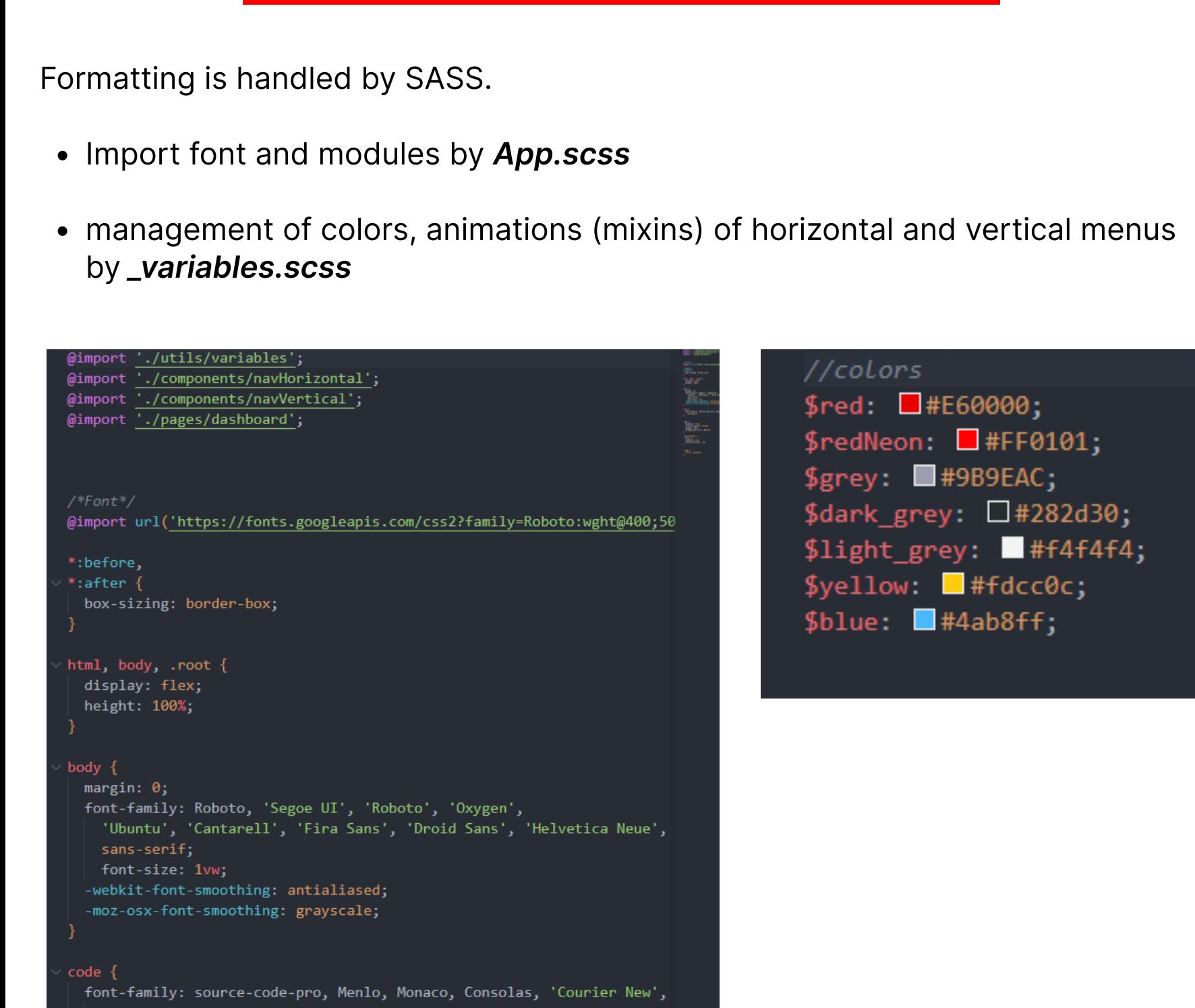
  const mockData = [
    USER_MAIN_DATA: {USER_MAIN_DATA, USER_ACTIVITY: USER_ACTIVITY, USER_AVG_SESSIONS: USER_AVERAGE_SESSIONS, USER_PERFORMANCE: USER_PERFORMANCE}
  ]

  const apiUrl = process.env[REACT_APP_API_URL]
  const apitarget = process.env[REACT_APP_${requestTarget}].replace('/:userId', userId)
  let usersData, data

  try {
    if (localStorage.getItem(`sportsee-${userId}-${requestTarget}`)) {
      console.log('Data is taken from your localStorage')
      data = JSON.parse(localStorage.getItem(`sportsee-${userId}-${requestTarget}`))
    } else {
      //console.log('Data is taken from $mockedEnv === "true" ? "MOCKED_DATA" : "Backend database with Axios".')
      if (mockedEnv === "true") {
        usersData = await new Promise((resolve) => resolve(mockData[requestTarget]))
        data = {
          ...usersData,
          filter: user => user.id === userId ? user.id === userId === userId : user.id === userId[0]
        }
      } else {
        try {
          usersData = await axios({
            method: 'get',
            baseURL: apiUrl,
            url: apitarget,
            responseType: 'stream'
          })
          data = usersData.data
          localStorage.setItem(`sportsee-${userId}-${requestTarget}`, JSON.stringify(data))
        } catch (error) {
          console.log(`Error axios: ${error}`)
        }
      }
    }
  } catch (error) {
    console.log(`Error ${requestTarget}: ${error}`)
  }
}
```

life cycle and asynchronicity

Representation of the life cycle of a component, as well as the process of retrieving the data required by each component(s) concerned.
Axios allows us to retrieve the data, the use of these implies an asynchronous management of the calls



using `useEffect()` to update the State involves using an array of dependencies

graphics

Creating charts with **Rechart JS**, a library for React JS.

It requires the import of components.

These components are placed in the **return** of the function component, in order to obtain a rendering.

These Recharts components have **props** and **payloads**, which allow chart customization.

```
return(
  <div className="Height">
    <dates &&
      <ResponsiveContainer width="100%" height="100%">
        <BarChart>
          <Bar>
            data={data}
            barCategoryGap={10}
            barCap={5}
            barSize={7}
            margin={{ top: 15, right: 15, left: 23, bottom: 15, }}
          </Bar>
        <CartesianGrid strokeDasharray="2 2" vertical={true} data={data} tickLine="day" tickLabelFormatter={(formatDate)} tickMargin={20} orientation="right" />
        <YAxis yaxisId="kilogram" type="number" domain={[ "dataMin - 3", "dataMax + 3" ]} tickLine="false" tickMargin={20} orientation="right" />
        <YAxis yaxisId="calories" type="number" domain={[ "dataMin - 50", "dataMax + 50" ]} hide="true" />
        <Legend verticalAlign="top" align="right" iconType="circle" iconSize={8} formatter={setLegend} />
        <Tooltip content={CustomTooltip} animationEasing="ease-out" active="true" />
        <Bar>
          yaxisId="kilogram" radius={20, 20, 0, 0} fill="#color:dark_grey"
        </Bar>
        <Bar>
          yaxisId="calories" radius={20, 20, 0, 0} fill="#color:#red"
        </Bar>
      </ResponsiveContainer>
  </div>
)

return(
  <div className="Score">
    <ResponsiveContainer width="100%" height="100%">
      <PieChart margin={{ top: 10, bottom: 10 }}>
        <text dy="50%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          ${userScore * 100}%
        </text>
        <text dy="60%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          de votre
        </text>
        <text dy="70%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          objectif
        </text>
        <Pie data={modelData} dataKey="value" nameKey="name" cx="50%" cy="50%" innerRadius="40%" outerRadius="60%" fill="#color:red" startAngle={180} endAngle={540} />
      <Legend verticalAlign="top" align="left" content={{ payload }}>
        <div style={{ color: "black", margin: "10px 0", opacity: ".8", position: "absolute", top: "0", left: "0" }}>
          ${payload.payload[0].value} ${payload.payload[0].value.substring(1)}
        </div>
      </Legend>
    </ResponsiveContainer>
  </div>
)
```

Prototypes

Prototypes allows verification of the type of parameters provided to components, in order to limit the risk of error, especially during team development on the same project.

```
//Prototypes
Counter.propTypes = {
  data: PropTypes.arrayOf(
    PropTypes.oneOfType([
      PropTypes.oneOf([
        "calorieCount",
        "proteinCount",
        "carbohydrateCount",
        "lipidCount",
      ]),
      PropTypes.number
    ])
  ).isRequired,
  i: PropTypes.string.isRequired,
};
```

```
//Prototypes
Duration.propTypes = {
  userId: PropTypes.number.isRequired,
  color: PropTypes.object.isRequired
};
```

SASS

Formatting is handled by SASS.

- Import font and modules by `App.scss`
- management of colors, animations (mixins) of horizontal and vertical menus by `_variables.scss`

```
@import './utils/variables';
@import './components/navHorizontal';
@import './components/navVertical';
@import './pages/dashboard';

/*Font*/
@font-face {
  font-family: 'Roboto', 'Segoe UI', 'Roboto', 'Oxygen',
  'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
  sans-serif;
  font-size: 1em;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

body {
  margin: 0;
  font-family: Roboto, 'Segoe UI', 'Roboto', 'Oxygen',
  'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
  sans-serif;
  font-size: 1em;
  height: 100%;
  background-color: #white;
}

.app-content {
  flex: 1;
  display: flex;
  flex-direction: row;
}

.logo {
  cursor: pointer;
}
```

```
//colors
$red: #E60000;
$redNeon: #FF0101;
$grey: #9B9EAC;
$dark_grey: #282d30;
$light_grey: #f4f4f4;
$yellow: #fdcc0c;
$blue: #4ab8ff;
```

Documentation and Readme

Comments have been placed on each function to allow good maintenance of the code.

The naming of the variables was done in a clear way with a precise notation system (PascalCase, snake_case, camelCase).

The code is documented according to established conventions.

- a cartouche presents the important functions, their parameter(s) and their return
- definition of the **type** for each parameter discussed
- use of english

```
return(
  <div className="Height">
    <dates &&
      <ResponsiveContainer width="100%" height="100%">
        <BarChart>
          <Bar>
            data={data}
            barCategoryGap={10}
            barCap={5}
            barSize={7}
            margin={{ top: 15, right: 15, left: 23, bottom: 15, }}
          </Bar>
        <CartesianGrid strokeDasharray="2 2" vertical={true} data={data} tickLine="day" tickLabelFormatter={(formatDate)} tickMargin={20} orientation="right" />
        <YAxis yaxisId="kilogram" type="number" domain={[ "dataMin - 3", "dataMax + 3" ]} tickLine="false" tickMargin={20} orientation="right" />
        <YAxis yaxisId="calories" type="number" domain={[ "dataMin - 50", "dataMax + 50" ]} hide="true" />
        <Legend verticalAlign="top" align="right" iconType="circle" iconSize={8} formatter={setLegend} />
        <Tooltip content={CustomTooltip} animationEasing="ease-out" active="true" />
        <Bar>
          yaxisId="kilogram" radius={20, 20, 0, 0} fill="#color:dark_grey"
        </Bar>
        <Bar>
          yaxisId="calories" radius={20, 20, 0, 0} fill="#color:#red"
        </Bar>
      </ResponsiveContainer>
  </div>
)

return(
  <div className="Score">
    <ResponsiveContainer width="100%" height="100%">
      <PieChart margin={{ top: 10, bottom: 10 }}>
        <text dy="50%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          ${userScore * 100}%
        </text>
        <text dy="60%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          de votre
        </text>
        <text dy="70%" dx="50%" textAnchor="middle" fill="#color:grey" style={{ fontSize: "2.2vw" }}>
          objectif
        </text>
        <Pie data={modelData} dataKey="value" nameKey="name" cx="50%" cy="50%" innerRadius="40%" outerRadius="60%" fill="#color:red" startAngle={180} endAngle={540} />
      <Legend verticalAlign="top" align="left" content={{ payload }}>
        <div style={{ color: "black", margin: "10px 0", opacity: ".8", position: "absolute", top: "0", left: "0" }}>
          ${payload.payload[0].value} ${payload.payload[0].value.substring(1)}
        </div>
      </Legend>
    </ResponsiveContainer>
  </div>
)
```