Programmation Web
Rapport Client-Side

Martin BOUTEILLER
GitHub : mbouteiller

27/02/2022

Taches effectuées

Lors de ce projet je me suis focalisé sur la partie frontend et j'ai notamment implémenté les composants de Tableau et son gestionnaire de données appelé MetaTab (qui s'occupe aussi de la récupération des données du composant Graph), j'ai ajouté la gestion des thèmes (clair / sombre) dans l'application et enfin j'ai fait l'affichage de l'adresse de la station sélectionnée.

Pour ce qui est de l'accès aux données, nous passons par une API REST qui permet de contacter le backend fait en Nest JS.

Stratégie de gestion de version avec Git

Notre stratégie de gestion de version employée pour ce projet est de type Git Flow. Nous avons donc une branche master et une branche develop. Dans notre cas, nous avons qu'une version mise sur la branche master, et celle-ci correspond à la version du projet que nous avons présenté lors de l'évaluation. Étant donnée la complexité du projet relativement faible, il n'était pas nécessaire de mettre en place un système plus élaboré. En effet, puisque les projets front et back sont placés dans des répertoires différents nous avons pu avancer chaque binôme de son coté sans impliquer de merges lors de nos commits.

Solutions choisies

Pour l'affichage de tableau j'ai utilisé ReactTable, une librairie qui présente les fonctionnalités que nous recherchions pour notre affichage de données sous forme de table et qui mets à disposition des exemples et des explications afin de faciliter la prise en main.

Pour la gestion du thème j'ai bien sûr utilisé le contexte pour pouvoir le passer facilement au sein de l'application, en revanche les composants Tab et Graph utilisent des props qui sont envoyés par MetaTab, leur récupérateur de données, puisqu'il n'y a qu'un seul niveau de parentalité pour ces composants et que l'objet à passer est uniquement un tableau de données, cela m'a paru plus adapté.

Difficultés rencontrées

La principale difficulté que j'ai rencontrée est la prise en main de la librairie React qui ne m'étais pas familière mais avec les cours j'ai pu découvrir des concepts très utiles tels que les hooks et la gestion des props et du context qui m'ont permis de débloquer certains points compliqués dans mon implémentation.

Temps de développement

Le développement de mes parties dans le frontend m'ont pris environ 6 heures de travail auxquelles il faut ajouter le temps de recherche de documentation et explications, trouvées sur Internet ou reçues en cours.

Code

```
// Render the UI for your table
return (
  <>
    <table {...getTableProps()}>
      <thead>
      {headerGroups.map(headerGroup => (
        <tr {...headerGroup.getHeaderGroupProps()}>
          {headerGroup.headers.map(column => (
            // Add the sorting props to control sorting. For this example
            // we can add them into the header props
            <th {...column.getHeaderProps(column.getSortByToggleProps())}>
              {column.render('Header')}
              {/* Add a sort direction indicator */}
              <span>
                  {column.isSorted
                    ? column.isSortedDesc
                      ? ' ▼'
                      : ' ▲'
                    : ''}
                </span>
            </th>
          ))}
        </tr>
      ))}
      </thead>
      <tbody {...getTableBodyProps()}>
      {page.map((row, i) => {
        prepareRow(row)
        return (
          <tr {...row.getRowProps()}>
            {row.cells.map(cell => {
              return <td {...cell.getCellProps()}>{cell.render('Cell')}</td>
            })}
          </tr>
        )
      })}
      </tbody>
    </table>
```

Cette capture représente le composant renvoyé par le fichier Tab.js qui présente en plus une partie de pagination à la suite. Ce composant est majoritairement implémenté grâce à des exemples de code pris sur les exemples de documentation de la librairie ReactTable. J'ai agencé ces morceaux de code afin de faire un tableau correspondant à nos besoins : l'affichages données que l'on souhaite, autoriser le tri de celles-ci et implémenter une pagination pour veiller à garder un affichage lisible. Ce composant est optimal pour moi car il est fait de parties basiques bien fusionnées pour créer un composant plus complexe et fonctionnel.

```
function makeChart(cell) {
  getStationPrices(
    cell.row.original.latitude,
    cell.row.original.longitude,
    cell.row.original.postalCode
  ).then((result) => {setPrices(result)});

  getAddress(
    cell.row.original.latitude,
    cell.row.original.longitude
  ).then((result :any[] ) => {setaddress(result)})
}

async function getStationPrices(lat, long, postalCode) {...}

async function getAddress(lat, long) {...}

return (
  <>
    <TabStyles>
      <Tab columns={columns} data={stations} />
    </TabStyles>
    <ChartStyles>
      <Chart data={prices} address={address} />
    </ChartStyles>
  </>
);
```

Ce code est le cœur du composant MetaTab qui gère la récupération des données pour le composant Tab et le composant Chart. Il est pratique mais n'est clairement pas adapté pour un projet plus ambitieux car le fait de centraliser la récupération de données dans le cas où l'on ajouterait plus de composants pourrait devenir un problème. De plus, le style des composants est géré directement dans le code métier grâce à la librairie styled-components ce qui est encore une fois une facilité d'implémentation mais devrait être séparé dans des fichiers css.