Programmation Web

Rapport Client-Side

Identifiant Github

LOSCIALE Vivian SI5

26/02/2022

Tâches effectuées

Dans notre équipe, nous nous sommes séparés en deux équipes de deux. Pour mon cas, j'ai été affecté à la partie serveur. Cependant, j'ai pu faire un peu de front avant le rendu final. Concernant la partie front, je me suis occupé uniquement du composant de graph, qui permet d'afficher la un graphique contenant des courbes de tout les carburant d'une station donnée, afin d'analyser son évolution au fil du temps.

Pour la partie serveur, nous avons pu mettre en place différentes fonctionnalités afin que le front puisse communique avec le serveur pour récupérer les données. Ainsi, la liaison entre le serveur et le front se fait de la façon suivante :

Le front dialogue avec le serveur via une API REST, en utilisant des requêtes HTTP.Il existe différentes routes pour accéder à certaines fonctionnalités, tels que la liste de tout les carburants disponibles dans la database, etc.

Pour faire une recherche spécifique, le front utilise une route spécifique en mettant dans le corps de la requête la liste des filtres que l'utilisateur à choisi. Afin de valider les données que le front nous envoie, nous avons mis en place dans le serveur des DTO (data transfer object) couplé à une librairie de validation des données (class-validator).

Dans le DTO d'entrée, nous avons définie tous les attributs possibles que le front peut nous envoyer, ainsi que la nature de chaque attribut et des contraintes sur celle-ci, tel que le fait qu'un attribut doit être un int supérieur à 0. Ainsi, si le front nous envoie une donnée erronée, une erreur sera renvoyée au front en spécifiant le problème.

Nous avons aussi des DTO de sortie, qui permettent de transformer les données afin qu'elles soient plus accessibles pour le front (notamment des renommages d'attribut et d'enlever des données inutiles pour le front).

Stratégie employée pour la gestion des versions avec Git

Concernant la stratégie employée pour ce projet, nous avons opté pour un branching strategy de type Git Flow. Nous avons donc une branche master et une branche develop. Lorsqu'une version nous satisfait, elle sera mise sur la branch master. Dans notre cas, nous aurons eu 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.

Solutions choisies

Le composant développé utilise la librairie « Recharts » qui permet de créer des graphiques assez simplement sur react. Nous avons choisi d'utiliser une librairie afin d'économiser du temps et de ne pas avoir à refaire tout depuis le début.

Il existe cependant d'autres librairies comme D3.js pour créer des graphes mais nous avons choisi « Recharts » par souci de simplicité.

Difficultés rencontrées

Comme toute librairies, il a fallu se pencher sur la documentation pour comprendre son fonctionnement. Après la lecture, la mise en place se fait de façon assez rapide. L'un des problèmes que j'ai pu avoir fut au niveau des informations à afficher sur le graph. Comme le serveur nous envoie un objet station contenant différentes données, il a fallu les trier afin de récupérer seulement celles qui nous intéressait, c'est-à-dire le prix de chaque carburant à différentes dates.

Temps de développement de la tâche

Pour le front, le temps effectué pour faire le composant de graph a été aux alentour d'une heure et demie.

Pour la partie serveur, le temps pour pouvoir créer les routes et écrire la logique pour les différentes fonctions doit être aux alentours de 5-6h.

Code

Étant donné que je n'ai pu faire qu'un seul composant au niveau du front, je vais exprimer ce qui semble optimal et ce qui mérite une amélioration. Vous pourrez trouver ci-dessous le composant en question.

```
1  import {LineChart, Legend, Line, Tooltip, XAxis, YAxis, CartesianGrid} from "recharts";
2
3  function Chart({data}) {
4
5    let formattedPrices = [];
6    let dates = [];
7
8    data.forEach((element) => {
9      if (!dates.includes(element.maj)) {
10       dates.push(element.maj)
11     }
12   })
13
14   dates.forEach((date) => {
15     let filteredFuel = data.filter((element) => {
16       return date === element.maj;
17     })
18
19     let result = {date: date};
20
21     filteredFuel.forEach((element) => {
22       result[element.nom] = element.valeur
23     })
24
25     formattedPrices.push(result);
26   })
27
28   console.log(formattedPrices)
29
30   return (
31     <>
32       <LineChart width={730} height={250} data={formattedPrices}
33                  margin={{ top: 5, right: 30, left: 20, bottom: 5 }}>
34         <CartesianGrid strokeDasharray="3 3" />
35         <XAxis dataKey="date" />
36         <YAxis />
37         <Tooltip />
38         <Legend />
39         <Line type="monotone" dataKey="E10" stroke="#8884d8" />
40         <Line type="monotone" dataKey="E85" stroke="#8884d8" />
41         <Line type="monotone" dataKey="GPLc" stroke="#8884d8" />
42         <Line type="monotone" dataKey="Gazole" stroke="#8884d8" />
43         <Line type="monotone" dataKey="SP95" stroke="#8884d8" />
44         <Line type="monotone" dataKey="SP98" stroke="#8884d8" />
45       </LineChart>
46     </>
47   )
48 }
49
50 export default Chart
```

Comme ce composant est majoritairement un composant de logique, la recherche des données semble optimale. On parcourt les données et on stocke les données à chaque étape, ce qui permet d'éviter de faire des vérifications non nécessaires. Ensuite, on formatte les données récupérées via l'objet pour les injecter dans le tableau via l'attribut data.

Un point d'amélioration serait entre la ligne 39 et 44, qui correspond aux tracés des différents carburants. Tel quel, il existe 6 carburants qu'on a mis en dur, mais il faudrait créer les lignes selon les carburant disponibles dans la station en question.
Si une station ne contient pas de GPLc, dans le cas actuel, on affichera toujours l'information de la courbe (bien qu'elle soit vide) dans le graph, alors qu'elle est inutile. Il faudrait lister toutes les instances de carburants disponibles dans l'objet de la station passé dans les props afin d'afficher de façon dynamique les lignes dans le graphique.