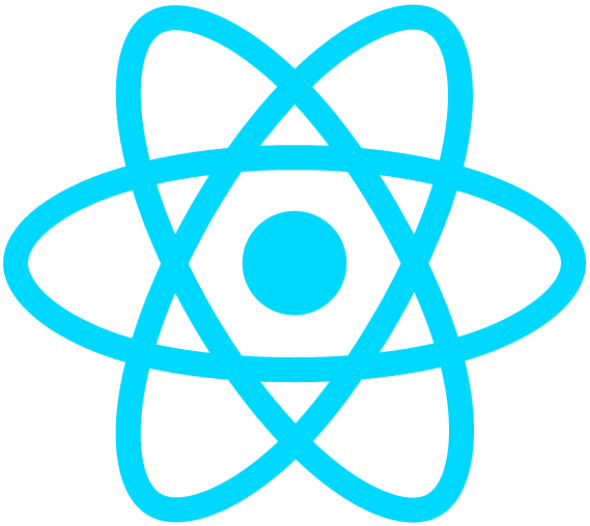
Rapport Projet
Front-end



Elise CHAMBERLIN
Github: https://github.com/Elise-Chamberlin

27/02/2022

I. Tâches effectuées

1. Mise en place d'un template

Dans un premier temps, lorsque le back-end n'était pas encore mis en place, j'ai créé un template de l'application contenant les éléments principaux. Le but était ici d'avoir des éléments facilement adaptables avec les futures données qui viennent du back-end. Ces composants ont ensuite été adaptés avec les véritables données.

Pour certains composants, BootStrap est utilisé. Il a aussi permis d'harmoniser le style de l'application.

Le template de base était composé d'une navbar, une barre de recherche pour recherche les stations essences dans une ville, les boutons pour filtrer le type d'essence voulue, et une Map.

Pour la Map, j'ai utilisé la librairie Leaflet (qui utilise les maps de OpenStreetMap) qui permet d'avoir une carte facilement ainsi que de placer des curseurs dessus en fonction de coordonnées GPS.

2. Tracé GPS

Dans la version actuelle de l'application, il est possible d'afficher le tracé du chemin à parcourir pour se rendre jusqu'à la station essence voulue. Pour cela, lorsque l'on clique sur une station sur la carte, l'on peut choisir de voir le chemin. Si la personne n'a pas accepté de partager sa position GPS, alors une alerte lui indique qu'il faut qu'elle accepte les autorisations. Sinon, le tracé s'affiche.

J'ai utilisé l'api de OpenRouteService qui permet de récupérer les informations pour réaliser un tracé GPS. Pour cela, il suffit de renseigner les coordonnées de départ et d'arrivée et l'API renvoie les étapes du chemin à parcourir. Les données sont ensuite stockées dans une liste et grâce à la librairie Leaflet, on peut utiliser le composant "Polyline" en lui passant la liste des étapes et cela trace le chemin sur la carte.

3. Loading Spinner

Lorsqu'il y a un appel au back-end pour récupérer des données, un spinner apparaît avec un fond blanc pour indiquer à l'utilisateur qu'un traitement est en cours et qu'il ne peut pas effectuer d'autres actions en attendant. Ce spinner animé est un composant Bootstrap. Il est mis par dessus tous les éléments de la page et prend 100% de la taille de l'écran. De cette façon, l'utilisateur comprend que son action a bien été prise en compte et qu'elle nécessite un temps de chargement. Lorsque l'on envoie une requête au backend alors le composant spinner passe à visible, et dès que l'on a une réponse du backend, le css de l'élément est remis à hidden.

4. Favoris

Les utilisateurs du site ont la possibilité d'avoir des stations essences favorites. Grâce à cela, ils peuvent consulter rapidement les prix de ces stations et leurs informations en allant dans la liste de leurs stations favorites accessible depuis l'onglet "Favoris" de la Navbar. L'utilisateur peut ajouter une station essence dans ses favoris en cliquant sur l'étoile vide présente dans le tableau des stations. Même principe pour enlever une station de ses favoris, l'utilisateur clique sur l'étoile pleine et cela l'enlève de sa liste.
Les stations essences favorites sont récupérées lors de la connexion au site avec un compte utilisateur et mises dans une variable dans le local storage. De cette façon, lorsque l'utilisateur veut afficher la liste de ses favoris, nous ne sommes pas obligés d'interroger le backend pour connaître les

stations favorites de l'utilisateur connecté. Ensuite, une requête est faite au backend pour récupérer les informations sur ces stations. De même dans la liste générale des stations, pour savoir si l'on doit afficher une étoile pleine ou une étoile vide, le local storage évite de faire une requête supplémentaire au backend.

Lorsque l'on clique sur l'étoile, une requête est envoyée au back (avec la route /user/addFavorite ou /user/removeFavorite) pour demander d'ajouter ou d'enlever la station des favoris. Mais si l'utilisateur cliquait de nouveau sur l'étoile de manière très rapprochée et que le back-end n'avait pas le temps de terminer sa requête, cela pose problème car de nombreuses requêtes inutiles sont effectuées. J'ai donc mis en place un spinner qui en plus d'indiquer que quelque chose est en train de charger, empêche l'utilisateur de pouvoir cliquer sur l'écran grâce à la div du spinner qui vient se placer au-dessus des éléments. Comme cela, il est impossible de renvoyer une autre requête entre temps et cela limite le nombre de requêtes qui sont faites.

5. Graphiques sur l'historique des prix de l'essence pour une station

Pour représenter l'historique des prix d'une station essence, j'ai utilisé la librairie Rechart qui permet de faire des graphiques. Nous pouvons visualiser sur les graphiques, l'évolution des prix des essences depuis 2021 jusqu'à aujourd'hui. Il y a donc trois courbes pour les trois types d'essence. Une pour le Gazole, une pour l'E10 et une autre pour le SP98. Les données historiques des prix sont récupérées pour une station, dont l'id est renseigné en props du composant permettant de faire les graphiques. Le composant appelle la route adaptée du backend (/querys/historyStation) avec comme paramètres l'id de la station et cela nous retourne une liste au format attendu pour l'afficher sous forme de graphique.

II. Gestion du Git

Pour chaque fonctionnalité principale, nous créons des issues Git et relions les commits à celles-ci. Nous avons un seul repository Github qui contient à la fois la partie front-end et la partie back-end de notre application qui sont séparées en deux dossiers. Nous avons travaillé sur une seule branche en s'assurant que celle-ci soit toujours une version stable de l'application

III. Solutions choisies

Pour afficher la map sur le site nous avons utilisé Leaflet. Leaflet est une librairie qui permet d'afficher une map mais aussi de la manipuler ce qui a été utile dans notre cas. En effet, Leaflet propose des composants qui permettent de tracer des routes très facilement. La librairie permet aussi de créer nos propres curseurs de map personnalisé et de les placer sur la carte simplement à l'aide de coordonnées GPS.

Leaflet utilise la carte de OpenStreetMap. Nous nous sommes tournés vers cette solution car elle est gratuite et a une communauté conséquente contrairement aux services de carte que Google propose (API Google Maps) mais qui sont payants.

De même pour la méthode qui permet de récupérer les étapes pour aller d'un point A à un point B en parcourant un chemin, j'ai utilisé un appel à l'api OpenRouteService qui dispose de nombreuses méthodes pour manipuler des coordonnées GPS et qui est gratuite contrairement à celle de Google.

Pour visualiser les données sous forme de graphiques, il existe plusieurs librairies permettant de faire ceci. J'ai choisi d'utiliser la librairie Rechart car cette librairie a été conçue spécialement pour React et dispose d'une communauté active. Il serait possible d'utiliser aussi d'autres librairies comme react-chartjs-2, Victory, … mais qui disposent d'une moins grande communauté.

Pour le composant graphique, j'ai fait le choix d'utiliser useEffect plutôt que de passer les données à l'aide des props. En effet, les graphiques sont affichés dans une fenêtre Canvas de Bootstrap qui contiennent aussi d'autres informations. Les autres informations en plus des graphiques sont déjà disponibles dans le Canvas car nous disposons de ces informations dans le composant parent. Mais pour l'historique des prix, nous avons besoin de faire un nouvel appel au backend et cela peut prendre quelques secondes. Ce composant graphique est utilisable seulement pour afficher l'historique des prix par essence, il faut donc obligatoirement lui passer une certaine liste avec les bons paramètres. Il n'était donc pas pratique de passer cette liste en props car il est fort possible qu'elle ne réponde pas au exigence du composant pour être affichée. On passe donc en props les éléments permettant d'identifier la station sur laquelle nous voulons faire un graphique et ensuite le composant se charge de récupérer et créer la liste adaptée à lui pour afficher les valeurs.

IV. Difficultés rencontrées

Rechart permet d'afficher des données sous forme de graphiques facilement à condition de lui donner les données dans le format qu'il attend. Seulement, les historiques des prix des stations ne sont pas toujours disponibles pour tous les types d'essence ou alors certaines dates manquent. Il y avait des problèmes d'affichage lorsque les données étaient incomplètes. Pour gérer cela, il a fallu faire un traitement dans le backend qui permet de trier et réorganiser la liste de la bonne manière pour les graphs. Il était plus intéressant de faire ce traitement dans le backend plutôt que dans le frontend car il y a pour certaines stations un grand nombre de données à envoyer. Comme ca on transmet seulement les données utiles à notre frontend.

V. Temps de développement par tâche

Chaque membre du groupe a à la fois travaillé sur la partie front-end et back-end de l'application. Les durée données sont donc le cumul du temps passé sur le front-end et le back-end.

Tâche	Durée
Mise en place d'un template avec les principaux composants	≃ 10 heures
Tracé GPS	≃ 2 heures
Favoris	≃ 24 heures
Graphiques	≃ 24 heures
Loading spinner	≃ 1h

VI. Commentaires de code

1. Composant “élégant”

```
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer } from 'recharts';
import {useEffect, useState} from "react";
import * as Utils from "../Utils";

const GraphPrice = (props) => {
    const [listPrices, setListPrices] = useState( initialState: []);
    useEffect( effect: () => {
        let request = JSON.stringify( value: { 'latitude': props.latitude,'longitude': props.longitude})
        Utils.default.sendRequest( methode: 'POST', url: '/querys/historyStation', request, callback: (res)=> {
            setListPrices(JSON.parse(res).listPrices)
        });
    }, deps: [props.latitude, props.longitude]);

    function CreateChart() {
       if(listPrices.length>4){
           return (
               <ResponsiveContainer width="100%" height="100%">
                   <LineChart
                       width={500}
                       height={300}
                       data={listPrices}
                       margin={{
                           top: 5,
                           right: 30,
                           left: 20,
                           bottom: 5,
                       }}
                   >
                       <CartesianGrid strokeDasharray="3 3" />
                       <XAxis dataKey="date" />
                       <YAxis domain={[1,2]}/>
                       <Tooltip />
                       <Legend />
                       <Line type="monotone" dataKey="Gazole" stroke="#8884d8" activeDot={{ r: 1 }} />
                       <Line type="monotone" dataKey="E10" stroke="#82ca9d" activeDot={{ r: 1 }} />
                       <Line type="monotone" dataKey="SP98" stroke="#f0ae91" activeDot={{ r: 1 }} />
                   </LineChart>
               </ResponsiveContainer>
           );
       }else{
           return (
               <div>L'historique des prix n'est pas disponible pour cette station</div>
           )
       }
    }

    return (
        <CreateChart/>
    );
};

export default GraphPrice;
```

Code du composant GraphPrice qui permet d’afficher l’historique des prix d’une station

Ce composant est le composant permettant de tracer un graphique de l’évolution des prix en fonction des types d’essences. Il est plutôt élégant car il a une unique fonction qui est de charger les données et de les afficher sous forme de graphiques. Il est donc très facilement réutilisable à

n'importe quel autre endroit de l'application ou alors carrément dans une autre application. Il récupère en props la longitude et la latitude de la station dont on veut l'historique des prix qui sert d'identifiant unique à la station. Lorsque le composant est Mounted alors il va faire un appel au backend (dans UseEffect) pour récupérer les données sur les historiques des prix grâce à l'identifiant donné en props. Le composant affiche soit un graphique si on a minimum 5 données, sinon on admet qu'un graphique ne serait pas très utile et on indique à la place que l'historique des prix n'est pas disponible pour la station concernée.

2. Composant "à revoir"

```
function addToFavoris(id) {
    if (!localStorage.getItem( key: "token")) {
        alert("Connectez-vous avant d'ajouter une station à vos favoris")
    } else {
        if (id !== undefined) {
            var storageStation = JSON.parse(localStorage.getItem( key: "favoriteStations"));
            storageStation.push(id);
            setFavoriteStationList(storageStation);
            Utils.default.sendRequest( methode: 'POST', url: '/user/addFavorite', JSON.stringify( value: {
                'idStation': id,
                'user': localStorage.getItem( key: "token")
            }), callback: ()=>{}).then(r => localStorage.setItem("favoriteStations", JSON.stringify(storageStation)))
        } else {
            alert("Impossible d'ajouter cette station aux favoris")
        }
    }
}
```

```
function removeToFavoris(id) {
    if (id !== undefined) {
        let storageStation = JSON.parse(localStorage.getItem( key: "favoriteStations"));
        let index = storageStation.indexOf(id);
        let firstPart = storageStation.splice(0, index);
        storageStation.shift()
        let secondPart = storageStation;
        storageStation = firstPart.concat(secondPart);
        setFavoriteStationList(storageStation);
        Utils.default.sendRequest( methode: 'POST', url: '/user/removeFavorite', JSON.stringify( value: { 'idStation': id, 'user': localStorage.getItem( key: "token") }), callback: () => {})
            .then(r => localStorage.setItem("favoriteStations", JSON.stringify(storageStation)))
    } else {
        alert("Impossible d'enlever cette station des favoris")
    }
}

function DisplayStar(props) {
    const idStation = props.coord1 + "," + props.coord2;
    const favStationList = localStorage.getItem( key: "favoriteStations");
    if (favStationList !== null) {
        if (favoriteStationList.includes(idStation)) {
            return <div id="favoriStar" onClick={() => removeToFavoris(idStation)}><FontAwesomeIcon
                icon={faStarSolid} /></div>;
        }
    }
    return <div id="favoriStar" onClick={() => { addToFavoris(idStation) }}><FontAwesomeIcon icon={faStarRegular} /></div>;
}
```

Code du composant ListEssenceFav pour gérer les favoris

Il aurait été plus intéressant de créer un composant unique pour l'étoile qui permet d'ajouter ou d'enlever une station essence de ses favoris. En effet, le code pour faire ces actions est dans les composants tableaux. Il y a donc duplication de code et de plus, si l'on souhaite ajouter la possibilité d'ajouter ou d'enlever une station essence de ses favoris à un autre endroit du site nous sommes encore obligés de dupliquer ce code-là. Le mettre dans un composant à lui tout seul avec comme props l'id de la station auquel il est associé aurait été un meilleur choix.