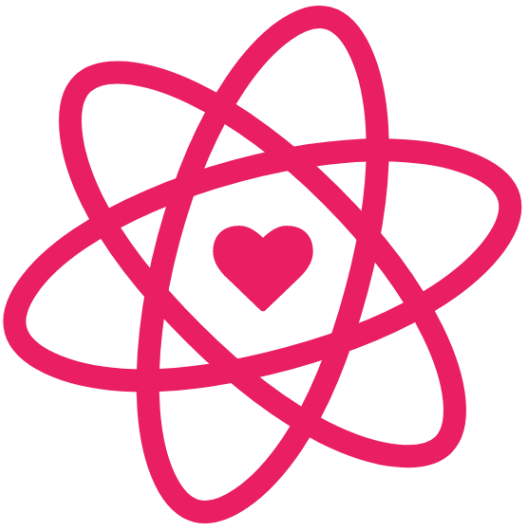
Yann Martin D'Escrienne
M2 Informatique (AL)



Rapport Individuel du projet React

Équipe :

- Quentin Larose
- Tigran Nersissian
- Yohann Tognetti
- Yann Martin D'Escrienne

Url du projet : https://github.com/SI5-QTYY-Web/si5-prog-web-project

Identifiant Github : Mentra20

I. Tâches effectuées

Voici ci-dessous la liste des tâches que j'ai effectuées côté frontend. Certaines partie de ces tâches ont pu être commencées, finalisées voir corrigées/stabilisées par un autre membre de l'équipe, mais je me suis occupé de la plus grande partie de chacunes.

- Création du composant GlobalMap : Ce composant est le composant qui contient la carte.
- Création des composants MapMarker : Ces composants correspondent aux icônes des stations sur la carte. Je les place en fonction de la position récupérée par le backend. Je me suis également occupé de la popup d'information qui apparaît en hover de ceux-ci.
- Création du composant StationDetailed : Ce composant correspond à la vue qui est affichée lorsque l'on veut les détails d'une station précise.

- Création de FilterBar et gestion du filtre sur les types d'essence. Ce composant correspond à la barre de filtre qui peut être affichée afin de filtrer les stations essences sur la carte.
- Création du composant NavBar correspond à la barre de navigation de l'application à l'aide de React Router.
- D'une manière générale, je me suis également occupée de la première mise en place de la communication entre le backend et le frontend sur la récupération des stations essences sur le filtre donnée (ou par défaut si première connexion). Mais également sur la récupération de plus d'information pour une stations essences lorsque cela est demandé (affichage du détail d'une station).
- Création du composant FavStationMenu (et FavStationMenuElement) correspond au station mis en favoris par l'utilisateur.
- Je me suis occupé de la mise en place du bouton afin de revenir à la position actuelle de l'utilisateur ou à la position par défaut si celui-ci n'a pas activé la géolocalisation. (via le composant MapTool)
- J'ai également créé et fait évoluer le composant StationList (et StationListElement) qui correspond à vue sous forme de liste des différentes stations affichées sur la carte.
- Je me suis occupé de la mise en place du tri par ordre alphabétique dans la liste des stations.
- D'une manière générale j'ai largement contribué au styling de la page via SCSS (toutes pages confondues excepté chartPage) ainsi que le correctif de problème aussi bien backend que frontend.

II. Stratégie employée pour la gestion des versions avec Git

Pour la stratégie employé pour la gestion de version avec git, l'équipe entière à travailler avec un développement sur chaque branche avec test manuel sur chacun avant fusion avec notre de développement principale (dev). Nous nous sommes répartis les tâches via Discord (La Covid en a forgée une utilisation à but scolaire).

III. Solutions choisies

Pour tout ce qui est affichage dynamique d'une carte, nous sommes parties sur l'utilisation de **React LeafLet**. Une alternative aurait été l'utilisation de Google Map. Ceci était un choix purement personnel mais dans les deux cas la configuration nous était suffisante.

Au sein du code react, lors d'un état partagé entre plusieurs composants (comme l'utilisateur connecté) nous somme partis sur des **useContext** afin d'éviter de surcharger App.tsx notamment en faisant remonter et redescendre tout les props qui commençaient à devenir assez nombreux. Dans le cas d'un état local à un composant ou deux, nous avons utilisé un **useState** classique. Pour tout ce qui est chargement des données à l'initialisation ou à des changement de valeur (comme un nouveau filtre) nous avons bien sur utiliser les **useEffect** de react.

Pour certains composants comme le slider ou un sélecteur avec recherche (comme celui dans la barre de filtre pour les services) nous avons utilisé des petits projets react fournissant ces composants hautement personnalisables. Par exemple pour le slider **rc-slider** et pour le composant select **react-select**. Il aurait été trop long de créer ces composants, parfois complexes, à la main ou d'utiliser ceux natif du html. Nous avons également essayé de mettre en place BootStrap mais cela n'est pas allé très loin.

Pour les requêtes vers le backend nous sommes partis sur **axios** car il s'agit du plus populaire en ce moment et qui répond largement à notre besoin.

IV. Difficultés rencontrées

Lors d'un stade assez peu avancé de notre projet, nous nous sommes heurtés à un problème de responsabilité sur App.tsx notre composant parent. Celui-ci comportait beaucoup trop de state (position actuelle, station affichées, la liste des services, des types d'essences….) car nous faisions descendre les states dans les composants fils en leur donnant un handler pour les modifier. Afin de régler ce problème nous avons redistribuer au mieux la responsabilité (avec des state locaux) et pour les données devant absolument être partagé, nous avons utilisé des contextes de react. Cela nous à permis d'avoir un code beaucoup plus clair et d'identifier rapidement les responsabilités de chacun de nos composants.

Personnellement, j'ai rencontré quelques difficultés concernant la position de l'utilisateur et la position ou les données sont actuellement chargées. Notre premier context ne conservait qu'une seule valeur ce qui créait un problème lorsque l'utilisateur voulait revenir sur sa position après avoir charger des données ailleurs. Pour régler ce problème, il a suffi d'étendre le contexte afin d'avoir deux positions distinctes en faisant bien attention de ne pas trigger les useEffect de l'un en changeant la position de l'autre (car anciennement partagé).

La mise en place de certains éléments à des positions précises ou avec un aspect précis via le SCSS m'a également posé quelques difficultés mais cela à été réglé grâce à mon collège Yohann Tognetti maîtrisant bien mieux le langage et ses propriétés que moi.

V. Temps de développement par tâche

- GlobalMap : 4%
- MapMarker : 6%
- StationDetailed : 18%
- FilterBar : 5%
- NavBar: 2%
- Communication backend/frontend : 8%
- FavStationMenu (et FavStationMenuElement) : 5%
- MapTool (et recenter sur l'utilisateur) : 4%
- StationList (et StationListElement) : 12%
- Tri par ordre alphabétique dans la liste des stations : 3%
- Styling et correctifs : 33%

VI. Un composant élégant

```
import { Map } from 'leaflet';
import React, { useContext, useEffect, useState } from 'react'
import { GeolocalisationContext } from '../../context/GeolocalisationContext';
import { MapContext } from '../../context/MapContext';
import "./MapTool.scss"
export default function MapTool() {

    const {userPosition,searchPosition,setSearchPosition}=
useContext(GeolocalisationContext);
    const {map}:{map:Map} = useContext(MapContext);
    const [userMove, setUserMove] = useState(false);

    const onDragEnd = ()=>{
        setUserMove(true);
    };

    const onRecenterClick = ()=>{
        setUserMove(false);
        setSearchPosition(userPosition)
        map.setView([userPosition.lat, userPosition.lon],13);
    }
    const onRecenterInfo = ()=>{
        const posUpdate=map.getCenter()
        setSearchPosition({lat:posUpdate.lat,lon:posUpdate.lng})
    }

    useEffect(()=>{//map est null à l'instanciation
        if(map) {
            map.on("dragend",onDragEnd);
        }
    },[map])

    return (
        <div className="toolBar">
            <div className='map-button'>
                <button className='buttonStyle' onClick={onRecenterInfo}>Afficher les
stations de la zone</button>
                {userMove && <button className='buttonStyle'
onClick={onRecenterClick}>Recentrer sur ma position</button>}
            </div>
        </div>
    )
}
```

Un des composants les plus smart que j'ai pu faire peut être le composant MapTool. Pas forcément dans sa logique mais plutôt dans son découpage de la responsabilité. En effet,

celui-ci prend une partie de la responsabilité concernant le chargement d'infos sur la zone où l'utilisateur se trouve et le fait de pouvoir revenir à sa position actuelle. Il est court et concis.
Mais il faut savoir que de base, tout composant englobé par une map React Leaflet peut utiliser un contexte natif useMap, mais ce n'est pas le cas de ce composant. J'ai donc créer un contexte plus général de la carte qui créer la référence à la carte dès son initialisation

```
const {setMap}:{setMap:any} = useContext(MapContext);
```

```
= {false} scrollWheelZoom={false} whenCreated={setMap}>
```

C'est ce qui me permet alors de faire des actions comme map.setView qui ne sont normalement pas possible à l'extérieur du container de la map React LeaftLet.
J'attache également à cette même carte un handler sur le dragEnd qui à lieu lorsque l'utilisateur fini de se déplacer sur la carte. Ce handler me permet de proposer un bouton qui va lui permettre de revenir à sa position initiale.

VI. Un composant effrayant

```
return(
    <div className='stationDetailed'>
        <h1>{gasStationInfo?.address||"chargement..."}</h1>
        <h3>{gasStationInfo?.id||"chargement..."}</h3>

        <div className="toolBar">
          <button className='buttonStyle' onClick={(e)=>{onItineraireClick()}} >Calculer un
itineraire</button>
          {favoriteButton()}
        </div>

        <div className='subInfo'>
          <h2>Essences</h2>
          {gasStationInfo? gasStationInfo.prices.map((value) => {
            const priceText = value.gasType+" : "+value.price+"€";
            return (<p key={value.gasType}>{priceText}</p>)
          }):
          <TailSpin color="#063d44" width={60} height={60}/>}
        </div>

        <div className='subInfo'>
          <h2>Services disponibles</h2>
          <ul>
            {gasStationInfo? gasStationInfo.services.map((service) => {return (<li
key={service}>{service}</li>)})
            :<TailSpin color="#063d44" width={60} height={60}/>}
          </ul>
        </div>

        <div className='subInfo'>
```

```
          <h2>Horaire</h2>

            {gasStationInfo?
              ((gasStationInfo.schedules.length !== 0) ?
                (<ul>
                  {gasStationInfo.schedules.map((schedule) => {
                    const scheduleText = schedule.day + (schedule.openned? " ouvert ":"
fermé ");

                    return (
                      <li key={schedule.day}>
                        <p>{scheduleText}</p>
                        {schedule.hourSchedule && schedule.hourSchedule.map((hour)=>{
                          return (<p key={hour.openHour}>de {hour.openHour} à
{hour.closedHour}</p>);
                        })}
                      </li>)}
                  )}
                </ul>)
                : (<p>Pas d'informations disponibles</p>))
            :<TailSpin color="#063d44" width={60} height={60}/>}
          </div>

          <button className='buttonStyle' onClick={(e)=>{onBackClick()}} >{"<< Liste des
stations"}</button>
          <button className='buttonStyleRed'
onClick={(e)=>{onUserReportClick(gasStationInfo!)}} >Signaler un problème</button>
      </div>
      );
}
```

Ceci est pour moi le composant le plus effrayant du code que j'ai pu faire. Il s'agit du return de mon composant StationDetailed.
C'est pour moi un code de piètre qualité. On voit clairement ici qu'il faudrait créer de nombreux sous-composants plutôt qu'un gros bloc de JSX. Cela rend juste la méthode "render" peu lisible et on ne voit pas clairement le découpage des responsabilités. Il mériterait clairement une optimisation et réécriture en sous-composant avec le bon passage des props.
Cela permettrait également d'avoir moins de state et de useContext présent dans le composant qui est assez chargé :

```
import useAxiosAuth from "../../hooks/axios-auth";

export default function SideMenu() {
  const navigate = useNavigateNoUpdates()
  const [gasStationInfo,setGasStationInfo] = useState<GasStationInfo>();
  const {id} = useParams();
  const { user, favoriteStations, setFavoriteStations, isLogged}:{user:string,favoriteStations:GasStationInfo[],setFavoriteStatio
  const {map,navControl,setNavControl}:{map:Map,navControl:any,setNavControl:any} = useContext(MapContext);
  const {userPosition,searchPosition,setSearchPosition} = useContext(GeolocalisationContext)
  const axiosAuth = useAxiosAuth();
  function getStationInfo(stationId:string) {
    console.log("GET STATION INFO FOR "+stationId)

    axios.get(BACKEND_BASE_URL+STATION_INFO+"/"+stationId)
```