Rapport final

Identifiant github: mehdicherkaouii

Sommaire

1. **Tâches effectuées**

- Le header de l'application.
- Parser le fichier des stations qu'on reçoit depuis le backend.
- Afficher les stations sur la map (react-leaflet).
- Modal qui liste les détails d'une station (adresse, prix des carburants et services) lorsqu'on clique dessus.
- Page de connexion.
- Page de création de compte.
- Filtre de recherche par carburants et par services.
- Thème sombre/clair sur toute l'application.
- Router pour naviguer entre les différentes pages.
- Menu déroulant pour changer le mode d'affichage (map, liste, graphe).
- Adapter le css lorsqu'on est en thème sombre sur toute l'application.

2. **Stratégie employée pour la gestion des versions avec Git**

Nous avons utilisé git pour une meilleure organisation du projet. Chaque membre du groupe avance sur sa partie après la répartition des tâches et qu'on pousse le code qui marche dans une branche locale afin de s'assurer d'avoir toujours une version stable.

3. **Solutions choisies**

L'utilisation de Mui qui est une bibliothèque robuste et personnalisable elle propose une multitude de composants fondamentaux et avancés. Elle nous a permis de construire une belle interface.

Il existe d'autres bibliothèques comme Ant Design, Reat Bootstrap, Grommet, Rebass, Blueprint … Nous avons choisi Mui parce qu'elle offre une documentation bien détaillée avec plusieurs exemples. Facile à ajouter dans un projet il suffit de lancer cette commande :

npm install @mui/material @emotion/react @emotion/styled dans le terminal et le tour est joué. Elle est employée dans les UI de grandes entreprises comme Netflix ou Amazone.

Mui nous a été très utile, elle nous a aider à développer plus rapidement et concevoir une belle interface.

4. Difficultés rencontrées

J’ai rencontré un point de blocage au niveau de la map lorsqu'on change le thème de light à dark ou l’inverse toute l’application prend le thème qui a été appliqué sauf la map il faut la recharger pour voir le résultat mais elle ne change pas dynamiquement.

Conception: je passe à la map le mode appliqué qui est un booléen (vrai si je suis en thème sombre et faut en thème clair) la map met à jour un “state” avec le “TileLayer” correspondant (le TileLayer applique un thème à la carte). J’ai fait des recherches sur internet et j'ai trouvé des personnes qui ont rencontré le même problème. En fait le “TileLayer” ne render qu’une seule fois même si je mets à jour la carte. Après plusieurs recherches je n’ai pas trouvé une solution pertinente pour changer le style de la carte dynamiquement sans forcer le dom a se recharger qui n’est pas une solution optimale elle va juste rendre l’application plus lente.

La solution pour laquelle j’ai opté est de mettre par défaut une carte qui n’est pas très claire et pas très sombre. La carte par défaut sera appliquée pour le thème clair et lorsqu'on passe en thème sombre je joue avec le style de la map en lui appliquant une opacité de 60% ce qui rend la carte plus sombre.

5. Temps de développement / tâche

- Le header de l’application. (3h)
- Parser le fichier des stations qu’on reçoit depuis le backend. (5h)
- Afficher les stations sur la map (react-leaflet). (4h)
- Modal qui liste les détails d’une station (adresse, prix des carburants et services) lorsqu’on clique dessus. (1h - 2h)
- Page de connexion. (1h - 2h)
- Page de création de compte. (1h - 2h)
- Filtre de recherche par carburants et par services. (3h)
- Thème sombre/clair sur toute l’application. (2h)
- Router pour naviguer entre les différentes pages. (1h)
- Menu déroulant pour changer le mode d’affichage (map, liste, graphe). (1h)
- Adapter le css lorsqu’on est en thème sombre sur toute l’application. (3h)

6. **Code**
    a. **Commenter une fonction ou un composant (max 100 lignes) que vous avez écrit et qui vous semble élégant ou optimal**

```
18  useEffect(() => {
19
20    STATIONS = props.stations;
21    setStationList(STATIONS);
22    if(!props.onChange && !props.service) {
23      setStationList(STATIONS);
24    } else if (props.onChange) {
25      const filtredStations = stationList.filter( station => {
26        let flag = false;
27        station.prix.map( p => {
28          if(p.nom.includes(props.onChange)) flag = true;
29        })
30        return flag;
31      }
32      );
33      setStationList(filtredStations);
34    } else {
35      const filtredStations = stationList.filter( station => {
36        let flag = false;
37        if( station.services){
38          let services = station.services.service;
39          for(let i = 0; i < services.length; i++) {
40            if(services[i].includes(props.service)) {
41              flag = true;
42            }
43          }
44        }
45
46        return flag;
47
48      }
49      );
50      console.log(filtredStations);
```

Cette méthode permet de filtrer les stations par carburant ou par service. Le code est englobé dans un “useEffect” afin qu’il soit exécuté après que la map soit chargée.

Une fois cette dernière est chargée avec toutes les stations, je fais une copie du tableau des stations récupérées depuis le serveur et j’applique les filtres sur ce tableau ce qui rend l’utilisation des filtres dynamique.

b. Commenter une fonction ou un composant (max 100 lignes) que vous avez écrit et qui mériterait une optimisation ou amélioration

J'ai créé deux composants Header différents, un avec les filtres qui s'affiche au-dessus de la map et un autre MyHeader sans filtres utilisé par les autres composants (liste et graphe) parce que ces deux derniers ne proposent pas des filtres. Ce n'est pas optimal, j'aurais pu créer un seul composant header et ce dernier devra englober tous les autres composants. Concernant l'adaptation du header en fonction du composant, j'aurais dû modifier le style du header dynamiquement en fonction du composant qui est rendu.

```
return (
  <Router>
    <div className={`App ${storageMode ? 'dark' : 'light'}`}>
      <div class="Toggle">
        <ToggleModeNight
        onChange={handleChangeMode}
        mode={storageMode}
        />
      </div>
      <Switch>
        <Route path="/signIn" >
          <SignIn setUser={setUser}/>
        </Route>
        <Route path="/signUp" >
          <SignUp setUser={setUser}/>
        </Route>
        <Route path="/dataTable">
          <MyHeader mode={storageMode} />
          <CollapsibleTable parentToChild = {stationsMap}/>
        </Route>
        <Route path="/chart">
        <MyHeader mode={storageMode} />
          <BarChart dataFromParent = {stationsChart}/>
        </Route>
        <Route path="/">
          <Header mode={storageMode} stations={stationsMap} currentPosition={currentPosition} onChange={handleChange}/>
        </Route>
      </Switch>
    </div>
  </Router>
);
```