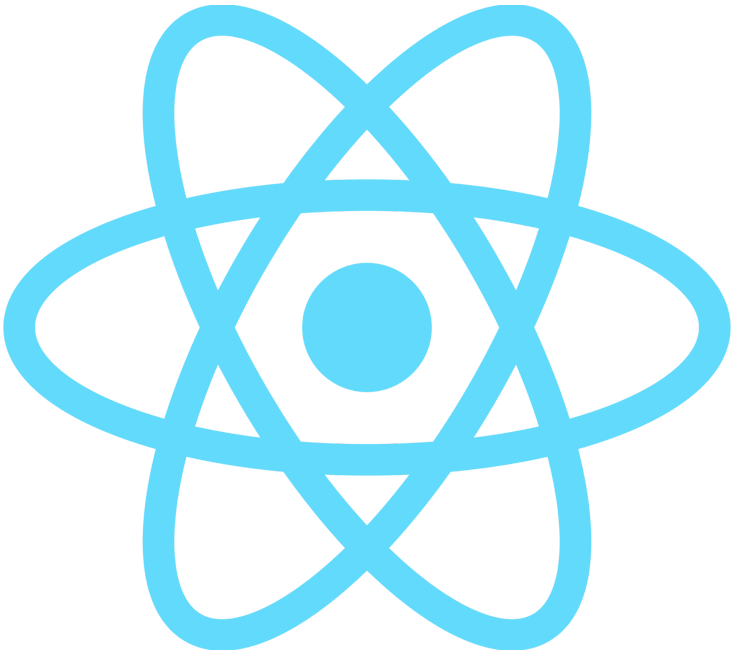
Yohann TOGNETTI
M2 Informatique (AL)



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 : Idrash -> https://github.com/Idrash

Table des matières

I. Tâches effectuées

Durant ce projet, j'ai effectué plusieurs tache clef dans le projet, en voici une liste :

- Mise en place du thème sombre : Pour le thème, j'ai mis en place un context "ThemeContext" qui retient si le thème est dark ou light et retient cette information dans le "local storage". Le projet étant parti sur du scss et non pas des styled component, le thème ajoute uniquement une classe dark-theme qui permet de faire des sélections correspondantes au thème.
- Mise en place des graphiques : J'ai utilisé la librairie "react-chartjs-2" pour mettre en place des graphiques. Ces graphiques utilisent des valeurs calculées par le backend et sont paramétrées pour utiliser le même bar de filtre. Cela permet d'avoir pour les mêmes stations afficher une autre représentation.
- Mise en place de spinner pour le chargement de la fenêtre de visualisation de la station en détail
- Gestion du css et du responsive : Je me suis occupé en majorité de la mise en place d'un thème responsive à l'aide de grid layout et quelque media query quand cela été nécessaire.
- Perfectionnement du scss, j'ai participé à la réalisation du scss et me suis principalement occupé de la résolution des différents problèmes d'affichages.
- Optimisation de la carte et mise en place de clusters. Nous avons eu des problèmes sur la carte du a un rafraichissement de l'intégralité de point sur la carte au moment du clic. J'ai utilisé le profiler de react pour trouver la raison de cette update puis modifier le "useNavigate" qui était source du problème sur react router v6. Celui-ci faisait un rafraîchissement à chaque modification de route.
- La transformation des filtres en "reducer" pour réduire la complexité et améliorer la maintenance des filtres qui sont communs à l'intégralité de l'application.
- Assistance pour mettre en place le routeur de l'application.
- Ajouter la possibilité d'avoir une station favorite. J'ai repris la tâche qui était déjà initiée pour aider à son implémentation qui avait quelque souci.
- Amélioration et correction de l'authentification. Notamment, la création d'un hook "useAxiosAuth" qui permet d'intégrer automatiquement les headers d'authentification et simplifier l'utilisation des requêtes qui nécessite d'être authentifié (notamment les favoris).
- Mise en place du composant checkbox liste qui nous a permis d'afficher une liste de checkbox. Cette tâche étant au début m'a aussi permis de prendre en mains react.

En plus de ces tâches, durant l'intégralité du projet, j'ai fortement aidé sur les différents problèmes rencontrés par les membres de mon équipe pour aider à les débloquer et leur expliquer les différentes erreurs.

II. Stratégie employée pour la gestion des versions avec Git

La gestion du projet est principalement faite avec Git ainsi que Discord. Dû à l'augmentation du distanciel, on a pris une aisance à travailler en asynchrone.

Au niveau de git, on a fait en sorte d'avoir notre branche dev qui possède toujours une version fonctionnelle de la version. De plus, on a fait en sorte d'ajouter des fonctionnalités petit à petit pour toujours avoir un livrable.

Pour implémenter les différents partis du frontend et du backend, nous avons mis en place un répertoire commun entre les deux qui possèdent des interfaces partagées. Cela nous a permis de nous mettre d'accord sur les différents échanges et être avec des données

III. Solutions choisies

Nous sommes partis sur une application qui a pour but d'être **compacte** avec une utilisation **implicite**. Dû à cela, nous avons fait des composants qui avaient pour but d'être utilisés pour les différentes pages de manière naturelle comme pour les filtres qui fonctionnent toujours de la même manière sur la liste, la carte et les graphiques. Cela a rajouté une difficulté car ces trois composants qui sont dans leurs nature différentes devaient fonctionner de manière similaire.

Pour ce qui est de l'affichage de la carte, nous avons opté pour **React Leaflet** qui est **Open Source**. Cela nous permet de ne pas dépendre d'un fournisseur comme Google et réduit le coût d'une éventuelle mise en production de l'application. Cependant, il est évident que la carte de Google Map a bien plus de fonctionnalités et aurait probablement été plus facile d'intégrer les différents comportements voulus. Finalement, nous avons réussi dans l'ensemble à tout de même avoir le résultat voulu et nous sommes ravis du rendu.

Au niveau de React, nous avons utilisé des librairies plutôt populaires pour faciliter la mise en place de notre application. On peut citer **React Router Dom** pour la gestion des routes de l'application, **Axios** pour la gestion des requête (ce choix a été fait pour rester sur la même technologie que sur le backend, le fetch native aurais pu répondre à notre besoin), **React Select** et **React Switch** pour des composants ou encore **React ChartJS 2** pour les graphiques.

En plus des différentes librairies, nous sommes partis sur une implémentation en **TypeScript** pour réduire le nombre d'erreurs de typage que nous aurions pu avoir tout au

long du projet. Avec cela, nous avons utilisé des **“React Functional Component”** pour implémenter nos différents composants étant bien plus léger que les classes. Au niveau des implémentation, nous avons utilisé different hook natif de React: **UseContext** afin d’éviter de surcharger App.tsx , **UseEffect** pour les différents événement du cycle des vis des composant, **UseState, UseMemo** et **useReducer** pour le reducer des filtres.

IV. Difficultés rencontrées

Vers le début de notre projet, ne connaissant pas trop React et ayant une application compacte, nous nous sommes retrouvés avec un fichier App.tsx qui avait des fonctions métier de plein de composants qui devait interagir entre eux. Cela a très vite augmenté la complexité du code. Grâce à votre aide, on a pu voir les différentes solutions pour résoudre ces problèmes. Notamment avec des **“context” et des “reducer”**. Cela nous a permis tout le long du projet de résoudre ce type de problème.

Ensuite, je me suis heurté à d'autres problèmes concernant la carte. Pour augmenter le nombre de points affichable sur la carte, j’ai mis en place des clusters qui sont fournis par React Leaflet. Une fois les clusters mis en place, le nombre de points supporter simultanément a fortement augmenté (la carte peut supporter d’avoir toutes les stations de France). Mais un gros problème de performance était visible au moment de cliquer sur un point pour afficher les informations détaillées dans la fenêtre à gauche. A l’aide du profiler, j’ai vu que tous les points sur la carte étaient chargés de nouveau à chaque fois que la fenêtre changeait. J’ai trouvé que cela etait du a useNavigate de **React Router Dom en V6** (cf. https://github.com/remix-run/react-router/issues/7634). Ce poste m’a permis de trouver une solution pour empêcher le chargement de l'état de “**useNavigate**” que j’ai implémenté dans le fichier “RouterUtils.tsx”. Cela m’a donc appris à utiliser différents outils pour résoudre les problèmes sur React.

V. Temps de développement par tâche

- Mise en place du thème sombre : 4%
- Resolution du probleme de useNavigate qui refresh tout : 10%
- Mise en place du CSS et responsive : 15%
- La transformation des filtres en “reducer”: 5%
- Mise en place d’un spinner sur la vue détaillée : 2%
- Mise en place des graphiques : 10%
- Assistance pour mettre en place le routeur de l’application : 1%
- Ajouter la possibilité d’avoir une station favorite : 7%
- Amélioration et correction de l’authentification : 5%
- Mise en place du composant checkbox liste : 6%
- Assistance de l'équipe sur les différents problèmes : 35%

VI. Un composant élégant

Le composant que je vais présenter est FilterStationContext. Ce composant n'est pas forcément le composant le plus technique que j'ai eu à mettre en place. Cependant je l'ai sélectionné car c'est le composant qui m'a le plus appris par le découplage des responsabilités qu'il apporte. Cela me permettra probablement par la suite de pouvoir créer de nouveau projet bien fondé en utilisant cette logique.

```
import {
   useReducer,
   createContext,
} from 'react';
import PropTypes from 'prop-types';

export const ADD_GAS_FILTER = 'ADD_GAS_FILTER';
export const SET_SERVICE_FILTER = 'ADD_SERVICE_FILTER';
export const REMOVE_GAS_FILTER = "REMOVE_GAS_FILTER"
export const UPDATE_RANGE_FILTER = "UPDATE_RANGE_FILTER"

export const RESET = 'RESET';

const initialState = {
   gasFilter: [],
   servicesFilter: [],
   range:20000
};
export const FilterStationContext = createContext();


const reducer = (state, action) => {
   let nextState = {};
   switch (action.type) {
   case UPDATE_RANGE_FILTER:
   nextState = {
       ...state,
       range:action.payload,
   };
   break;
   case ADD_GAS_FILTER:
       nextState = {
           ...state,
           gasFilter: [...state.gasFilter, action.payload],
       };
       break;
   case SET_SERVICE_FILTER:
       nextState = {
           ...state,
           servicesFilter: action.payload,
       };
       break;
```

```
    case REMOVE_GAS_FILTER:
    nextState = {
        ...state,
        gasFilter: state.gasFilter.filter((elt)=>elt!==action.payload),
    };
    break;
    case RESET:
        nextState = initialState;
        break;
    default:
        throw new Error();
    }
    console.log(nextState);
    return nextState;
};

export const FilterStationContextProvider = ({ children }) => {
    const { NODE_ENV } = process.env;
    const [state, dispatch] = useReducer(reducer,initialState);
    return (
        <FilterStationContext.Provider value={{ filterState:state, dispatch }} >
            {children}
        </FilterStationContext.Provider>
    );
};

FilterStationContextProvider.propTypes = {
    children: PropTypes.node.isRequired,
};
```

Comme vous pouvez le voir, il s'agit d'un “reducer” qui est situé dans un contexte. La logique métier de celui-ci est plutôt facile, on a un état qui est mis à jour à l'aide du reducer et d'une action. La fonction reducer prend en paramètre l'état courant et l'action à effectuer et renvoie l'état suivant (tout en prenant en compte qu'un état doit être immuable et donc une copie doit être effectuée). On utilise ensuite le hook useReducer de React qui nous permet d'avoir un couple [state,dispatcher] qui nous permet d'avoir un unique état qui est automatiquement pris en compte à l'appel de la fonction dispatch. Pour finir, le contexte nous permet d'accéder à l'état et au dispatcher facilement depuis l'intégralité des composants fils.
Cela permet vraiment d'avoir une forte modularité et me semble un excellent moyen de faire évoluer une application de manière durable, il est donc pour moi le composant le plus important à mettre en avant durant ce rapport malgré sa simplicité au niveau logique.

VII. Un composant pas élégant

Le composant le moins élégant que j'ai eu à modifier et dont je suis aussi responsable de l'évolution du à l'ajout de la gestion du chargement est “StationDetailed” qui s'occupe de l'affichage en détail d'une station.
Celui-ci possède beaucoup d'éléments avec des map ainsi que de l'affichage conditionnel. Du a cela, on se retrouve avec des blocs comme cela :

```
<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>
```

Il aurait été intéressant de regrouper les éléments comme celui-là dans des sous composants pour simplifier la lisibilité. Si j'avais eu plus de temps j'aurais effectué cette modification a l'ajout du spinner de chargement.
Cela aurait donné des composants avec une portée plus réduite comme un composant "services disponibles" qui aurait été géré par lui même s' il est en chargement ou non.