TIGRAN NERSISSIAN
M2 Informatique AL

UNIVERSITÉ CÔTE D'AZUR

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 : TIGRAN06 → https://github.com/TIGRAN06
Identifiant Github Secondaire : nt506155

I. Tâches effectuées

Durant ce projet, j'ai effectué plusieurs tâches clés. En voici une liste :

- **Mise en place du slider.**
  J'ai implémenté dans un premier temps le slider natif html, puis j'ai utilisé la librairie slider-rc pour implémenter le composant SliderReact chargé de changer la taille du rayon de recherche. Pour cela j'ai utilisé le dispatcher (reducer) en passant par FilterStationContext afin de partager la valeur du rayon avec les autres composants en charge de faire l'appel vers le backend pour récupérer la liste des stations.

- **Correction des graphiques.**
  J'ai utilisé FilterStationContext pour changer la taille du rayon de recherche des stations lorsqu'on est dans le menu "Graphiques".

- **Mise en place de la navigation.**
  J'ai mis en place la navigation GPS en ajoutant le bouton "calculer l'itinéraire" et en utilisant routing machine leafltet. J'ai aussi changé la langue de l'affichage du plan en français et le mode "car" en utilisant la documentation.

- **Création du bouton "Signaler un problème".**
  J'ai créé le bouton "Signaler un problème" afin d'afficher un prompt et demander à l'utilisateur de saisir un message puis de l'envoyer afin qu'il soit stocké dans la bd en faisant un appel vers le backend en cas de problème.

- **Mise en place de la communication du backend.**
  J'ai aussi mis en place la communication du backend avec le front, afin de créer des Objects utilisant les interfaces partagées dans common pour interfacer les données du backend selon le format attendu par le front.

- **Mise en place du 2dsphere dans mongodb.**
  J'ai mis en place le 2dsphere dans mongodb afin de gérer les coordonnées des stations sous format geoJson et de pouvoir utiliser des méthodes built-in optimisées pour la recherche de point dans un cercle sur un plan polaire. ($NearPoint,$MaxDistance etc..)

- **Mise en place d'un composant button et d'un composant select** chargés de trier la liste des stations selon le type d'essence choisie, du plus chère au moins chère, et inversement. J'ai utilisé la méthode built-in Array.sort() pour avoir une méthode de tri optimisée en termes de complexité en espace/temps.

- **Mise en place de l'affichage du cercle de rayon de recherche.**
  J'ai mis en place l'affichage du cercle de rayon de recherche (toutes les stations doivent être incluses dans ce cercle). Pour cela j'ai bien fait attention à effacer les layers que j'ai ajouté à la map intégrant les variables/function navControl, setNavControl dans MapContext. Ceci afin de ne pas dupliquer les cercles lorsqu'il y a un changement et que le useEffect est appelé pour mettre à jour stationList.

- **Création et mise en place de GasStationPositionContext.**
  J'ai créé et mis en place GasStationPositionContext afin de partager la liste des stations à travers les différents composants.

- J'ai fait **80% du MVP** (fetch en mode cron des données du gouv plus stockage dans bd mongo avec un schéma mongoose. Filtrer selon station type d'essence/service, map avec les markercluster en fonction de ce que le back renvoi et mise en place du 2dsphere dès le MVP).

- **Création et mise en place du bouton “Afficher les stations de la zone”.**
  J’ai créé le bouton permettant d’afficher les stations proches du drag-end de l’utilisateur ou de sa position courante ainsi que la position de la station sélectionnée.

En plus de ces tâches, durant l'intégralité du projet, j’ai aidé les membres de mon équipe avec les différents problèmes rencontrés. A débloquer ces problèmes ou pour être un support lorsque le scope qu’ils avaient choisi demandait plus de ressources que prévu initialement (en temps et énergie).

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, nous avons pris une aisance à travailler en asynchrone.

Au niveau de git, nous avons fait en sorte d’avoir notre branche dev qui possède toujours une version fonctionnelle. De plus, nous avons ajouté au fur et à mesure des fonctionnalités pour toujours avoir un livrable.

Pour implémenter les différentes parties 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 cohérents lors de l’échange 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 créé des composants pour les utiliser dans 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 différents de par leur nature, 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 il aurait été probablement plus facile d'intégrer les différents comportements voulus. Finalement, dans l’ensemble, nous avons tout de même réussi à obtenir le résultat voulu et nous sommes ravis du rendu.

Pour la navigation (plan d'itinéraire) nous avons utilisé **React Leafltet Routing Machine** afin d’afficher le chemin et le plan des itinéraires (instruction pour se rendre à destination). Étant une surcouche open source de React Leaflet, l'intégration de cette libraire dans notre projet s’est faite naturellement.

Au niveau de React, nous avons utilisé des librairies plutôt populaires pour faciliter la mise en place de notre application. Nous pouvons citer **React Router Dom** pour la gestion des routes de l’application, **Axios** pour la gestion des requêtes (ce choix a été fait pour rester sur la même technologie que sur le backend, le fetch native aurait pu répondre à notre besoin en utilisant un custom hook comme **useSWR,** qui d’ailleurs a été utilisé pour le **MVP**), **React Select** et **React Switch** pour des composants ou encore **React ChartJS 2** pour les graphiques.

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 à la main ces composants qui sont parfois complexes, ou d'utiliser ceux natifs du html.

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 légers que les classes. Au niveau des implémentations, nous avons utilisé différents hook natifs de React: **UseContext** afin d'éviter de surcharger App.tsx, **UseEffect** pour les différents événements du cycle de vie des composants, **UseState, useReducer** pour le reducer des filtres.

IV. Difficultés rencontrées

Dans les débuts de notre projet, en ne connaissant pas trop React et en ayant une application compacte, nous nous sommes retrouvés avec un fichier App.tsx qui avait des fonctions métier de plein de composants qui devaient interagir entre eux. Cela a très vite augmenté la complexité du code. Grâce à l'aide de mes camarades Yohann Tognetti et Yann Martin D'Escrienne j'ai eu la chance d'apprendre à utiliser les différentes solutions pour résoudre ces problèmes que nous avons vus en cours. Notamment avec des **"context" et des "reducer"** quand j'ai voulu rajouter le rayon de recherche dans le menu FiltreBar.
Cela m'a permis d'apprendre tout le long du projet comment résoudre ce type de problème.

Ensuite, je me suis heurté à d'autres problèmes concernant la carte. Je n'ai pas réussi à mettre en place un mécanisme pour fermer le plan d'itinéraire (hook,click). Il aurait fallu utiliser Leaflet-control-geocoder et ajouter une option geocoder dans L.Routing. Au départ le code était lent car il affichait tous les points et le client navigateur ne supportait pas la charge. Du coup, j'ai mis en place le rayon afin de limiter le nombre de stations chargées en cache. Aussi, j'ai mis en place un tri en utilisant la méthode built-in Array.sort(), autrement le tri n'aurait pas fonctionné car la méthode faite à la main prenait trop de temps pour s'exécuter (complexité O(n^3) ou n c'est la taille de la liste).

V. Temps de développement par tâche

- Mise en place des graphiques : 2%
- MapTool/MapPage (afficher les stations proches du drag-end de l'utilisateur ou de sa position courante) : 10%
- Station List (création et mise en place Context) : 5%
- Tri par prix croissant/décroissant selon le type d'essence choisie : 3%
- Styling et correctifs :5%
- GlobalMap : 5%
- MapMarker : 5%
- StationDetailed : 5%
- FilterBar : 5%
- addNavigation (Gps + plan d'itinéraire) : 10%
- Communication backend/frontend : 10%
- Slider-RC (création et implémentation du rayon de recherche en utilisant le dispatcher) 15%
- Backend : 20 %
-

VI. Des composants élégants

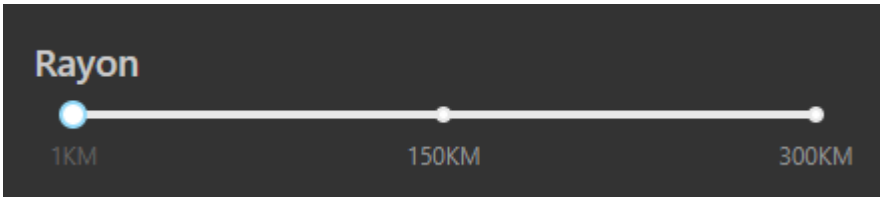
```
import Slider from 'rc-slider';
```

```
const createSliderWithTooltip = Slider.createSliderWithTooltip;
const Range = createSliderWithTooltip(Slider.Range);

export default function SliderReact({value, onSliderChange}:
  {value:number, onSliderChange: (value:number)=>void}) {

  return (
      <Range
      min={1000}
      max={300000}
      step={5000}
      onAfterChange={(value) => {
        onSliderChange(value[0]);
      } }
      defaultValue={[value]}
      marks={{1000:"1KM",150000:"150KM",300000:"300KM"}}
      tipFormatter={value => (value/1000)+"KM"}
      />
  )
}
```

Un des composants les plus simples, clairs et concis que j'ai créé est le slider. Il est très complet grâce au framework RC-slider car il y a plus d'options (tipFormatter, marks etc) que le slider natif et il est très simple à comprendre et à utiliser. Le petit bémol reste au niveau de l'héritage des propriétés css depuis la libraire RC-slider qui rendent le composant légèrement difficile à modifier.



J'ai aussi affiché le rayon sur la carte de façon fluide et j'ai fais attention à ce que l'appel vers le backend ne se fasse uniquement que lorsque l'utilisateur relâche son clic du slider. Ceci dans le but de ne pas spammer d'appel http le backend lorsque la valeur du rayon de recherche change.
Je suis également fier d'avoir pu intégrer la possibilité d'afficher l'itinéraire sur la carte en rouge et le détail du plan de voyage avec juste quelques lignes. Ceci en faisant attention de remove le layer précédent s'il existe afin de ne pas dupliquer l'affichage de l'itinéraire :

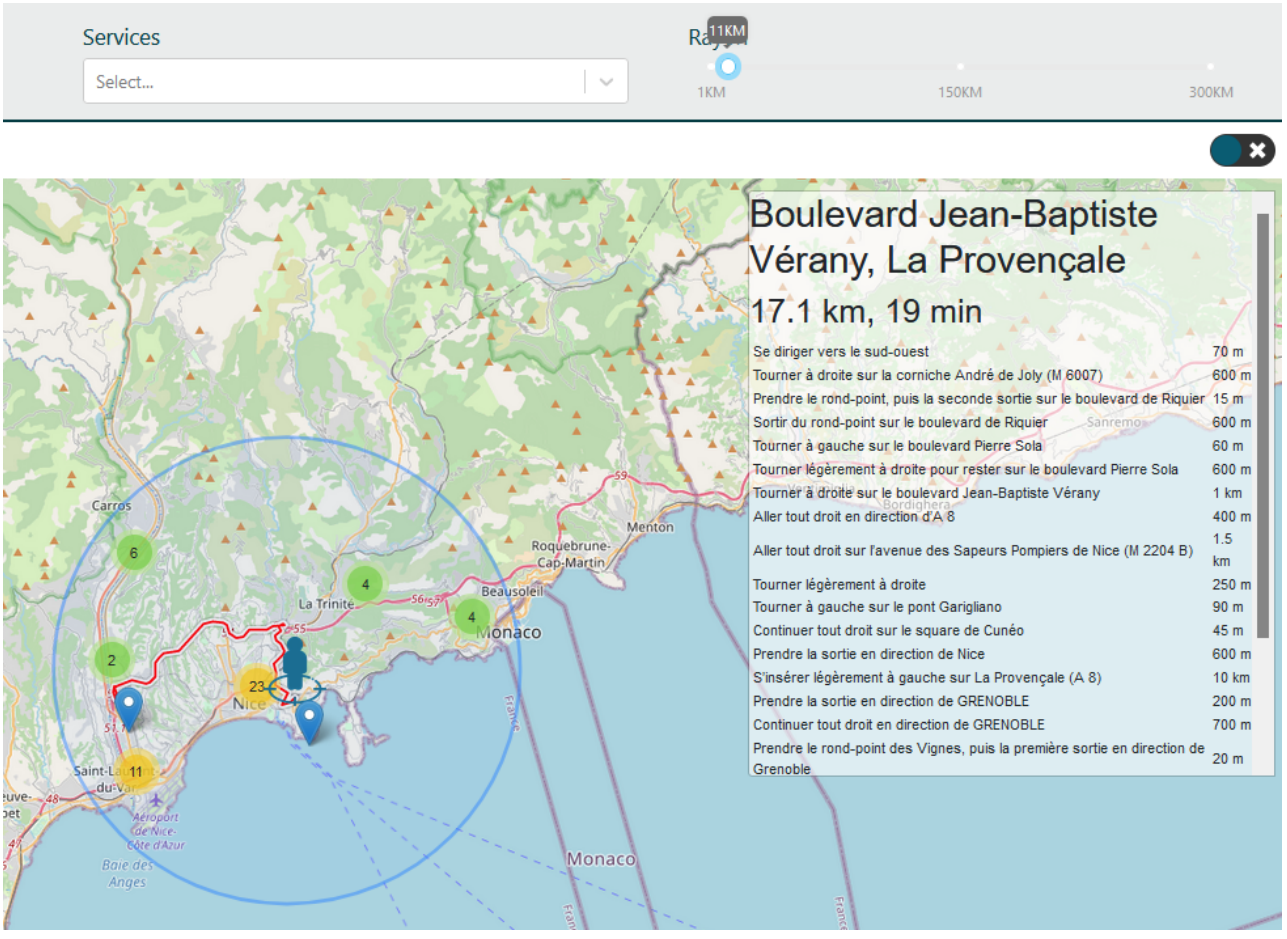
```
 function onItineraireClick(){
       addNavigation()
}

const addNavigation = () =>{
  if(navControl){
    map.removeControl(navControl)
  }
  setNavControl(L.Routing.control({
    routeWhileDragging: true,
    show:true,
```

```
      router: L.Routing.osrmv1({
        language: 'fr',
        profile: 'car'
      }),
      waypoints: [
          L.latLng(userPosition.lat, userPosition.lon),
          L.latLng(gasStationInfo?.position.lat||0,gasStationInfo?.position.lon||0)
      ]

    }).addTo(map)
    )
```

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



VI. Un composant effrayant

```
export default function StationList() {
  const [isAscending, setIsAscending] = useState(true);
```

```
  const [isPriceAscending, setIsPriceAscending] = useState(true);
  const [fuelList, setFuelList] = useState([])
  const [stationList, setStationList] = useContext(GasStationPositionContext)
  const [isLoad, setLoad] = useState(false)
  const { isLogged } = useContext(AuthContext)
  const navigate = useNavigateNoUpdates()

. . .

  const sortListClickByPrice = () => {
    let sortMethod = null;
    const stations = []
    const stationsWithoutTypeGas = []
    console.log(stationList)

    const selectedGas: string = (document.getElementById("selectPrice") as HTMLInputElement).value;
    for (const e of stationList) {
      let exist = false
      for (const gas of e.prices) {
        if (gas.gasType == selectedGas) {
          stations.push(e)
          console.log(gas.gasType == selectedGas)
          exist = true
          break
        }
      }
      if (!exist) {
        stationsWithoutTypeGas.push(e)
      }
    }

    stations.sort((a: GasStationPosition, b: GasStationPosition) => {
      isPriceAscending ? sortMethod = sortPriceUp : sortMethod = sortPriceDown
      return sortMethod(a, b);
    })

    setIsPriceAscending(!isPriceAscending);
    console.log(stations.length)
    setStationList([...stations, ...stationsWithoutTypeGas])

  }
 . . .
  return (
    <div className='stationList'>
      <h1>Stations Essences :</h1>

      {isLoad ?
        (<div>
          {isLogged && <button className="buttonStyle"
onClick={()=>navigate("/favoris")}>Favoris</button>}
```

```
          <button className="buttonStyle" onClick={sortListClickByAdress}>Trier par adresse</button>
          <button className="buttonStyle" onClick={sortListClickByPrice}>Trier par prix</button>
          <select id="selectPrice">
            {fuelList.map((elt) => { return <option value={elt}>{elt}</option> })}
          </select>
          {stationList?.map((station: GasStationPosition) => {
            return <StationListElement key={station.id} gasStation={station} />
          })}
        </div>)
        : (
          <div>
            <h2>Chargement...</h2>
            <TailSpin color="#063d44" width={60} height={60} />
          </div>)}
    </div>
  )
}
```

Ce composant est l'un des plus mauvais sur lequel j'ai pu travailler car il représente beaucoup de responsabilités. En effet, il pourrait être divisé en sous composant et aussi si nous avions eu le temps j'aurai pu faire un slider avec un onClickChange/onClick. Ceci afin de détecter si l'utilisateur choisit un type de station d'essence et que le menu puisse se trier en fonction, au lieu d'utiliser le select puis le bouton, ce qui complique un peu l'UX (2 clicks au lieu d'un).