Rapport – Projet React

Par Barthélemy Passin-Cauneau

(pseudo github : BarthelemyPassinCauneau)

I) Travail effectué

Après avoir travaillé sur la partie serveur (mise en place de la base de données distante MongoDB et de quelques endpoints basiques pour l'accès au back-end afin d'avoir une base de travail fonctionnelle), j'ai travaillé sur divers éléments qui constituent la partie client.

Je me suis occupé de :

- la recherche des données live du Covid en France et création de son composant pour les afficher (~6h)
- l'implémentation du switch permettant de changer entre le mode sombre et clair ainsi que de la quasi totalité des changements d'affichage qui en découlent (~9h)
- détection automatique du darkmode pour l'utilisateur en vérifiant ses paramètres par défaut (~2h)
- la localisation de l'utilisateur (~6h)
- la mise en place des Lazy imports (~4h)
- les aspects responsive (~3h)
- autre : refactor divers, nettoyage du code, home page et mise en page CSS (~6h)

En tout est pour tout, en plus des 8 heures de TD hebdomadaires j'ai dû travailler approximativement une dizaine d'heures sur la partie serveur (en comprenant MongoDB) et une vingtaine sur la partie client.

II) Gestion du projet

Le projet n'étant que sur deux semaines, nous n'avons pas déployé des versions différentes de notre logiciel sur git. Nous avons analysé les grandes parties à réaliser pour ce projet et sommes attribués les taches. Nous nous sommes contentés de créer une branche annexe Dev dans laquelle nous faisions nos changements et nous mettions sur Master la version finale du projet. Cependant nous avons utilisé des préfixes dans nos push (build, fix, refactor) afin de mieux visualiser les changements apportés par l'équipe.

III) Analyse du code

a) Fonctionnalités implémentées

La première fonctionnalité affichant le tableau des données live du Covid en France, s'exécute simplement grâce à un lazy import. Le composant principal App appelle le composant FetchFranceLiveGlobalData qui retourne les données dès qu'elle sont reçues. Le lazy import charge alors le composant Grid en lui passant les données en paramètre et le tableau est affiché. Cette architecture étant relativement simple, j'ai voulu avant tout séparer les composants en fonction de leur tache a réaliser. Ainsi le composant Grid se contente de retourner du HTML, le composant

Fetch se contente de récupérer les données et de les retourner et le composant App appelle ces fils et affiche le tout. Cette architecture permet de bien séparer les fonctionnalités entre les composants et sera retrouvée plusieurs fois dans les autres parties du projet.

Le système du darkmode a été inspiré du système implémenté dans Itunes API du cours. Un boolean stocke le mode d'affichage (true => dark mode, false => light mode). Ce boolean est alors récupéré par le composant principal App qui change le texte et le background en fonction de sa valeur. Ce boolean est cependant transmis à quelques composants fils (notamment la Map ou le Graph) car le changement de mode effectué dans le CSS de App ne concerne pas les balises spécifiques de ces derniers composants. Une solution alternative à ceci serait de faire en sorte que ce boolean puisse être une variable globale et donc accessible par tous les autres composants ou d'ajouter ces balises spécifiques dans le CSS de App.

La localisation de l'utilisateur reprend la même logique que pour afficher les données live. Le composant Graph ayant besoin de cette information (pour afficher les données du département de l'utilisateur à l'initialisation), App appelle le composant FetchLocationUserDep qui renvoie le département de l'utilisateur et import (via un lazy import) le composant.

Enfin, les aspects responsive du site se font uniquement via le CSS des composants concernés. N'ayant que peu d'éléments à afficher sur la page, j'ai décidé de déplacer le tableau des données live en bas de la page puis d'enlever le switch du darkmode quand la largeur de l'écran est trop petite. Les graphes s'adaptant automatiquement et la carte ayant une taille fixe, nous n'avions pas d'autres éléments à modifier.

b) Problèmes rencontrés

Pendant ce projet nous avons été confronté à quelques problèmes, l'un d'eux était l'affichage des composants alors que les données requises n'avaient pas encore été récupérées via les API.

En effet au début du projet le composant principal App faisait les différents appels au serveur et aux API pour récupérer les données tout en affichant les composants fils. Problème qui en découlait alors : les composants affichaient des erreurs à l'initialisation car les données n'avaient pas encore eu le temps d'être obtenues. Même si ce problème s'est rapidement résolu par une simple vérification de l'état des données au début du composant (data!=undefined ou encore data.length > 0), cette solution n'était pas du tout propre et ne nous convenait pas. Ainsi j'ai fait des recherches et découvert la fonctionnalité des Lazy imports ainsi que l'utilisation des balises Supense de React. En implémentant cette solution à notre projet, nous avons pu corriger ce problème plus proprement et alléger notre code. Cette partie est je pense, la partie la mieux réalisée que j'ai pu faire :

Dans App.js :

```
const Grid = lazy(
    () => new Promise((resolve, reject) =>
      FetchFranceLiveGlobalData.then(data => {
        setRealtimeData(data);
        resolve(import("./components/Grid"));
      })
    )
  );


…
<Suspense fallback={
             <div className="loadGrid">
                    <p className="loadingText">Accès aux données en direct en
cours...</p>
                    <div className="loader"></div>
```

```
            </div>}>
      <Grid data={realtimeData} mode={displayMode} />
</Suspense>
```

Dans Grid.js :

```
const Grid = ({data, mode}) => {
    return (
        <div className={`Grid ${mode ? 'dark' : 'light'}`}>
            <table>
                ...
            </table>
        </div>
    );
};

export default Grid;
```

Suite à la découverte des lazy imports, nous avons décidé de les implémenter dans tous les composants qui en avaient besoin et ainsi rendre notre projet plus propre.

Ensuite, un problème qui m'a concerné directement était l'utilisation du bout de code permettant de vérifier les préférences de l'utilisateur et d'appliquer automatiquement le darkmode. Ne comprenant pas exactement comment cela fonctionne, j'ai fait cette vérification dans un useEffect mais en y ajoutant un boolean qui est immédiatement mis à false pour simuler la fonction de componentDidMount() d'une classe. Même si cela apporte le comportement souhaité, de toutes les parties que j'ai pu faire il s'agit de la fonctionnalité dont je suis la moins satisfaite et qui mériterait amplement une amélioration afin de la rendre plus propre.

```
UseEffect(() => {
    if (init && window.matchMedia("(prefers-color-scheme: dark)").matches) {
      setDisplayMode(true);
      setInit(false);
    }
});
```

Un dernier problème que j'ai dû corriger sans vraiment savoir les autres alternatives qui pourraient s'offrir à moi était la localisation de l'utilisateur. Les données fournies par le navigateur ne sont qu'une latitude et longitude, et les données que nous affichons sur notre site sont en fonction des départements ou régions. Ainsi pour passer de ces coordonnées à un numéro de département, j'ai dû faire appel à une API externe pour récupérer l'adresse de l'utilisateur, rallongeant donc le temps pour déterminer le département de l'utilisateur. Même si cette partie a été codée proprement (grâce aux lazy imports), sa réalisation en revanche ne me satisfait pas et je pense que d'autres solutions plus faciles et moins coûteuses en temps auraient pu être envisagées.