RAPPORT PWS CLIENT SIDE

Maël DELABY

Identifiant github

MaelDelaby

Tâches effectuées / temps de développement

Récupération et utilisation des données utilisées dans les pages list, map et graph avec l'aide de Maël Vaillant—Beuchot : ~8h

Affichage du nombre de cas confirmés en France : ~1h30

Stratégie employée pour la gestion des versions avec Git

Nous avons travaillé de la façon suivante :

- Utilisation d'issues github afin de se répartir les tâches
- Utilisation d'un kanban afin de gérer le processus de développement d'une fonctionnalité
- Création d'une branche pour travailler sur une issue
- Utilisation des pulls requests devants être approuvées par un autre développeur
- Merge sur la branch dev

Solutions choisies

Un des choix les plus important et qui a été le plus compliqué pour moi a été le fait d'utiliser le framework MaterialUI car je ne l'avais que très peu utilisé auparavant. Nous avons choisi ce framework car il permettait de faciliter le développement en offrant des composants permettant d'afficher nos données comme souhaité (graph, list, map), de plus plusieurs de mes camarades le maitrisaient bien. Nous aurions pu utiliser d'autres framework comme React Bootstrap par exemple, mais l'expérience de mes camarades nous a fait choisir MaterialUI.

Comme dit dans le choix précèdent, la page graph contient un objet venant de MaterialUI permettant d'afficher le graph, ce dernier est <LineChart>, nous avons donc choisi ce composant car nous utilisions MaterialUI il était donc logique d'utiliser des components issus de ce dernier, <LineChart> était celui qui correspondait à notre problématique d'affichage de données temporelles. Nous aurions pu utiliser d'autres librairies permettant de faire ça comme Google Charts, mais nous aurions aussi pu extraire la responsabilité liée à la création du graphe en utilisant une API créant le graphe à notre place comme le fait très facilement quickchart.io, cela n'a pas été choisi car cela rajoute des dépendances à des API qui ne sont clairement pas utiles car nous pouvons créer facilement le graphe avec MaterialUI.

Le choix de l'API pour le nombre de cas en France s'est fait sur https://coronavirusapi-france.now.sh/, cette API permet de faire tout ce que nous voulions : avoir des informations simples d'utilisation, rafraichies tous les jours et provenant de sources sures car souvent gouvernementales. Cette API

s'appuyant sur des chiffres donnés par d'autres organismes nous aurions peut être pu trouver ces chiffres d'une autre façon, mais au moins si une sources ne marche plus cette API permettra toujours de nous trouver une autre source plutôt que ça soit à nous de chercher une nouvelle source.

Difficultés rencontrées

Les difficultés que j'ai rencontrées sont les suivantes :

- L'utilisation des composants MaterialUI car je n'avais que très peu utilisé ce framework auparavant, ça a été une découverte pour moi que j'ai mis du temps à maitriser
- React de façon générale car je ne maitrise pas très bien cette technologie et d'autres membres du groupe la maitrisaient très bien ce qui m'incitait à avoir du code construit de la bonne façon, cela m'a pris du temps pour m'améliorer et j'ai donc perdu du temps sur des choses très simples pour moi avec d'autres technologies
- Le plantage de mon ordinateur 2 jours avant le rendu, donc grosse source de stress, pertes de commits importants et perte de beaucoup de temps pour trouver un autre ordinateur utilisable

Code

J'ai décidé que mon composant élégant et mon composant à optimiser seraient le même car ayant travaillé pour une grosse partie de mes composant avec Maël Vaillant—Beuchot, j'aimerais présenter un composant que j'ai fait tout seul pour le composant élégant, mais comme je pense qu'on pourrait l'améliorer j'ai décidé de choisir ce dernier pour le composant à optimiser. En soit il n'a pas besoin dans l'immédiat d'être optimiser mais il pourrait l'être selon l'évolution de l'API qu'il utilise, ce composant est donc le composant ShowHowManyCase.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { makeStyles } from '@material-ui/core/styles';

const useStyles = makeStyles((theme) => ({
    root: {
      position: 'absolute',
      bottom: 0,
      width: '97%',
      textAlign: 'right',
    },

}));

export default function ShowHowManyCase() {
    const classes = useStyles();

    const [nbOfCase, setNbOfCase] = useState([]);

    const urlForFrance = "https://coronavirusapi-france.now.sh/FranceLiveGlobalData";
```

```
    const timeoutToRefesh = 1000;

    async function refresh(){
        try {
            const data = await axios.get(urlForFrance);
            return data.data.FranceGlobalLiveData[0].gueris + data.data.FranceGlobalLiveData[0].deces;
        } catch (error) {
            console.log(error);
        }
    }

    useEffect(() => {
        const fetchData = async () => {
            setNbOfCase(await refresh());
        };
        fetchData();
        let interval = setInterval(async () => fetchData(), timeoutToRefesh);
        return () => {
            clearInterval(interval);
        };
    }, []);


    return (
        <div className={classes.root}>
            <p>
                {
                    nbOfCase > 0 ?
                        'Nombre de cas confirmés en france : ' + nbOfCase :
                        'Requete à l\'API coronavirusapi-france.now.sh impossible'
                }
            </p>
        </div>
    );
}
```

Composant élégant

Ce composant est élégant car en quelques lignes grâce à makeStyles, nous avons un composant qui peut être utilisé sur toute les pages en fournissant une information très importante : le nombre de de cas confirmés.

De plus ce composant utilise diverses choses intéressantes tel que useState, qui permet de rafraîchir le component lors de la modification du nombre de cas, useEffect qui permet d'exécuter quelque chose après chaque affichage ainsi que setInterval qui permet de lancer régulièrement une fonction ce qui nous permet de mettre régulièrement à jour notre composant.

Composant à optimiser

Nous pourrions imaginer que l'API que ce composant utilise n'arrive plus à accéder aux données de openCOVID19-fr mais accède aux données d'une autre source, nous n'aurions peut-être alors pas la même structure de données et donc peut être pas le nombre de morts et de guéris, il faudrait alors changer le code de ce composant. Pour combler ce problème on pourrait chercher tous les attributs qui pourrait nous aider à déterminer le nombre de cas en France et afficher le résultat plutôt que de n'afficher le résultat que si nous arrivons à trouver le nombre de mort et de guéris.