Rapport Programmable Web Client Side

Identifiant github: RayanBenmammar

Tâches effectuées:

- Component BarPlot, histogramme permettant d'afficher le nombre total de tests effectuées ou le nombre total de cas recensés par régions depuis le début de l'épidémie (15h)
- Header (2h)
- Bouton dans le component RateComponent permettant d'afficher en modal le texte explicatif (30min)
- CSS en général (4h)
- fetch les données de la veille depuis une api externe (1h)
- loading spinners au niveau du fetch des données dans Barplot et des données de la veille dans le header (2h)

Stratégie employée pour la gestion du git:

Nous avons mis en place un modèle de branching assez courant, qui lie chaque feature avec une branche spécifique portant le nom de la feature. Cette manière de faire nous a permis de pouvoir développer chacun notre travail dans notre coin sans être impacté par les modifications des autres membres.
Une fois notre travail terminé sur nos branches respectives, on effectuait un merge de nos branches par rapport à notre branche commune develop (unreviewed-Work dans notre cas), ceci afin de pouvoir régler les différents conflits.
Et lorsque nous avions sur develop un ensemble de fonctionnalités qui marchaient et qui constituaient une brique importante de la solution finale, alors on merge develop sur la branche main.
De cette manière on s'assure qu'à n'importe quel moment la version qui existe sur main est fonctionnelle, et que si jamais au dernier moment un gros problème se passe sur notre code nous avons une version non complète mais fonctionnelle à montrer.

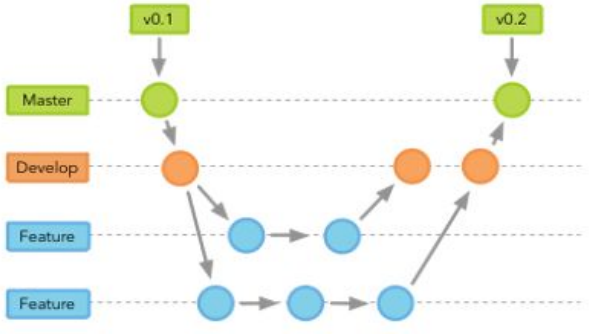

Figure 1 : Exemple de stratégie de branching utilisée

Difficultés rencontrées:

- Utilisation correcte du UseEffect

J'ai dû relire la doc pour pouvoir avoir correctement 2 des utilités du UseEffect qui correspondent au ComponentDidMount et au ComponentDidUpdate.
En effet, dans mon composant Barplot, je devais faire en sorte que le graphique se charge seulement après la récupération des données, et se change lors du changement de valeur du select. Il m'a fallu donc revoir l'utilisation des "dependencies" associées à ce hook.

- Difficultés avec d3js en général

L'intégration et surtout la modification du composant venant de la doc d3js ont été compliquées à mettre en place, car la doc était faite pour du Javascript Vanilla, et certaines fonctionnalités sur React ne fonctionnaient pas comme voulu.
Une difficulté qui persiste toujours, c'est que les données sur l'axe de Y ne s'affichent pas. On peut les voir dans le code HTML produit, mais elles ne sont pas affichées. Pour pallier ce problème, j'ai fait l'utilisation de tooltip pour afficher ces données manquantes.

De même pour le merge des données affichées.
En effet, lorsque l'on change la valeur du select présent, cela permet de changer les données à afficher sur le graphique, et malgré avoir fait exactement comme dans la documentation, au lieu de s'effacer pour laisser apparaître les nouvelles données, les données qui étaient présentes restaient affichées et le graphique était inutilisable.
Le problème a été résolu après maintes et maintes tests différents, mais cela m'a fait perdre pas mal de temps.

- Difficulté au niveau du git

Pas une difficulté au niveau du code en lui-même, mais plutôt au niveau du git.
On a eu des problèmes de merge qui nous ont pris pas mal de temps et qui ne nous a pas permis de faire l'entièreté des fonctionnalités que nous voulions implémenter.

Code :

```jsx
export const ChartRegions = () => {
    const selectData = ['cas', 'tests'];
    const [selected, setSelect] = useState(selectData[1]);
    const [data, setData] = useState( initialState: []);
    const [loading, setLoading] = useState( initialState: true);
    const url = 'http://localhost:9428/api/serviceregions/classe0';

    const fetchData = async () => {
        const response = await fetch(url);
        const json = await response.json();
        setData(json);
        setLoading( value: false);
    }

    useEffect( effect: () => {
        fetchData().then( value => {
            setSelect(selectData[0]);
        });
    }, deps: []);

    return (
        <div style={{paddingTop:"40px"}}>
            {loading && <Loader type="Bars" color="#00BFFF" height={40} width={40}/> }
            {!loading && <div>
                <BarPlot
                    select={selected} data={data}
                />
                <form>
                    <label>
                        Select :</label>
                    <select value={selected} onChange={e => {
                        setSelect(e.target.value);
                        // handleChange();
                    }}>
                        {selectData.map((d : string ) => (
                            <option key={d} value={d}>{d}</option>
                        ))}
                    </select>
                </form>
            </div>}
        </div>
    );
```

Je pense que mon composant ChartRegions est assez bien correct.
Il utilise bien les hooks en général, avec UseEffect et UseState.
Il me permet à la fois d'envoyer les données au composant Barplot après les avoir récupérées depuis le Back-End de par la fonction asynchrone "fetchData" qui permet de "set" les données à envoyer au composant, et aussi de modifier les données à

afficher dans ce même composant en faisant changer la variable “select” depuis une balise “select”.
De plus, j’ai fait l’utilisation d’un loader qui permet à l’utilisateur que des données sont en train d’être récupérées, et que l’affichage se fera par la suite.

```
useEffect( effect: () => {

    if( props.data.length > 0) {
        const data = props.data;
        data.sort( (a,b) => {
            return a[props.select] < b[props.select];
        })
        console.table(data);

        // append the svg object to the body of the page
        svg.append("svg")
            .attr( name: "width", value: width + margin.left + margin.right)
            .attr( name: "height", value: height + margin.top + margin.bottom)

        svg.selectAll( selector: "*").remove();
        // Initialize the X axis
        let x = d3.scaleLinear()
        .range( range: [1, width])

        let xAxis = svg.append("g")
            .attr( name: "transform", value: "translate(0," + height + ")")


        // Initialize the Y axis
        let y = d3.scaleBand()
            .range( range: [ height, 0])

        let yAxis = svg.append("g")
            .attr( name: 'width', value: 200)
            .attr( name: "class", value: "myYaxis")
```

Dans ce composant qui correspond au Barplot, il y a quelques lignes de code qui permettent de faire fonctionner l’app mais qui pourraient être plus optimales j’estime.

En effet, pour pouvoir attendre de tracer le graphique seulement à la réception des données récupérées depuis le back, une condition qui vérifie l’attribut length du tableau de données est placée. Ce qu’il aurait fallu réussir à faire c’est de ne pas avoir ce code au niveau du render initial du composant mais seulement dans le useEffect qui détectait le changement de valeur de ce tableau de données.

De plus, l’initialisation des axes X et Y devrait se faire à l’initialisation du composant et modifiée seulement après lors des changements de valeur, et ne pas comme ici les supprimer et les initialiser de nouveau.

Malheureusement je n’ai pas réussi à mettre cela en place, j’ai donc laissé ce code fonctionnel à la place.