Projet web client-side

Identifiant github

ColombanThomas

Tâches effectuées

- Composant *Location :* Gère l'affichage du bouton « Localisez-moi » et récupère les coordonnées GPS du l'utilisateur.
- Composant *Map* : Gère l'affichage de la Map nationale et régionale, et du tooltip lors d'un hover sur une région/ un département.

Utilisation de Git

Nous avons mis en place un model 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 mon travail terminé sur ma branche, j'effectuais un rebase de ma branche par rapport a notre branche commune develop, ceci afin de pouvoir régler les différents conflits sur ma branche et non sur develop. Une fois ma branche mise à jour avec develop je fais une merge request si la fonctionnalité que je viens de développer est importante et mérite d'être relue par un autre membre de l'équipe. Si c'est un petit développement je merge moi-même directement ma branche avec develop.

Et lorsque nous avons sur develop un ensemble de fonctionnalités qui marchent et qui constituent une brique importante de la solution finale, alors on merge develop sur la branche main.

De cette manière on s'assure qu'a 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 a vous montrer.

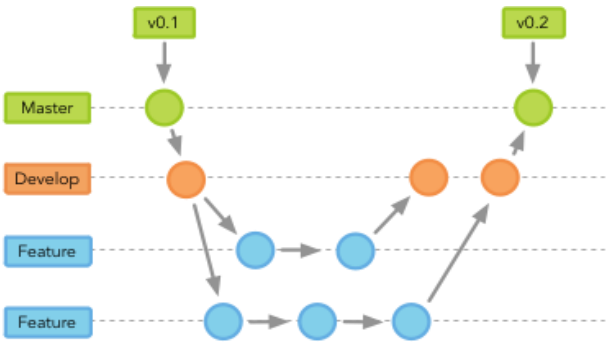

Figure 1 : Exemple de stratégie de branching utilisée

Map

Pour le composant Map qui s'occupe de l'affichage des différentes Map de l'application j'ai choisi d'utiliser un Hook, c'était un choix qui m'a paru séduisant au début car le composant que je souhaitais créer répondait à toutes les caractéristiques du Hook :

- Mon composant n'avait pas besoin d'état local il doit seulement gérer l'affichage d'un élément <svg> et il y a très peu d'interaction avec cet élément (changement de page au click).
- La Map s'affiche en une fois lors du chargement de la page et donc pas besoin d'avoir des méthodes comme componentDidMount, etc pour gérer les effets de bord.

De plus mon composant ne devait même pas gérer ce qu'il devait afficher car je lui passais en paramètre le fichier geojson (fichier qui décrit point par point la Map) à afficher.

Mais au fur et a mesure du développement de l'entièreté du composant je me suis aperçu qu'il y avait des cas particuliers lors qu'on souhaite afficher la Map de la France ou la Map d'une région.

Les informations affichées dans le tooltip ne sont pas les mêmes, le click sur les éléments du <svg> ne redirigent pas au même endroit, etc…

J'ai donc opté pour la solution de créer 2 composants différents, un pour la Map de la France, et l'autre pour la Map des régions. Mais en finissant de développer cette solution je me suis aperçu qu'une grande partie des 2 composants était semblables, et qu'il y avait énormément de duplication de code.

J'ai donc choisi de retourner sur la solution initiale avec un seul composant qui s'occupe de l'affichage des 2 <svg> différents. Même si cela m'obligeait à mettre à certains endroits des conditions car en fonction du type de Map qui je voulais le processus était différent.

Ce n'était pour moi pas dérangeant car cela restait anecdotique et minimal.

Mais lors de l'intégration de ce composant avec les autres composants des autres membres de l'équipe, des conditions se sont rajoutées à mon composant Map pour faire face aux différents cas d'utilisation.

Lors aussi de l'intégration de mon second composant qui gère la localisation j'ai dû ajouter des effets de bord à mon composant Map (l'utilisateur clique sur le bouton pour se localiser, ce qui va ajouter un cercle rouge sur la Map qui montre sa position, ça oblique donc le composant Map se re-render).

De même l'ajout du tooltip sur la Map force le re-render car nous allons avoir un rendu différent à chaque fois que l'utilisateur bouge sa souris.

Avec donc ces deux fonctionnalités et les effets de bord qu'elles engendrent, j'ai dû utiliser le Hook d'effet dans mon composant Map, pour pouvoir gérer toutes ces mises à jour d'informations sur le <svg>.

Il s'est aussi posé le problème des performances quant aux appels récurrents de mon composant vers le backend : a chaque fois que la souris passe sur un département ou une région on affiche des informations relatives à ce département/région qui viennent directement de la base de données. Mais je me suis rendu compte du problème que ça pouvait engendrer lorsqu'un utilisateur bouge sa souris très rapidement sur l'ensemble de la Map : une requête est faite en direction du backend pour

chaque survol, et comme la valeur n'est jamais stockée niveau frontend on refait plusieurs fois les mêmes appels.

J'ai donc essayé de régler ce problème en stockant via des Hook d'état ces données dans le front, mais j'ai rencontré énormément de soucis à cause de l'asynchronicité des Hook d'état. Par manque de temps pour me pencher sur la question j'ai choisi de garder une requête pour chaque changement d'élément au niveau du hover pour pouvoir livrer un composant fonctionnel au rendu, même si je suis bien conscient que cette solution n'est ni optimale ni performante.

En prenant du recul sur mon code je vois bien que ce que j'ai fait n'était pas le plus approprié pour les besoins que j'avais. Si je pense toujours que l'utilisation d'un Hook était une bonne idée au vu de mes besoins au départ, ces besoins ont évolués au cours du développement du projet et la solution que j'ai choisi au début et qui m'a semblée adaptée ne l'est plus du tout au vu des besoins finaux et de tout ce que ça implique.

Je pense donc que c'est ce composant qui mériterait une optimisation et une amélioration.

Avec plus de temps pour rendre le projet j'aurais fait un refactor complet du composant, en utilisant non plus un Hook mais bel et bien une classe Component.

Ce choix me paraît, avec du recul et au vu de ce que fait mon composant, le plus approprié.

Car j'ai choisi au début d'utiliser un Hook car je n'avais pas besoin d'avoir des états locals et des effets de bord. Mais j'ai rajouté ça à mon Hook au cours du développement.

Je ferais un composant sous forme de classe pour pouvoir bénéficier des méthodes qui gèrent les effets de bord tel que l'affichage du point rouge de localisation, ce qui éviterait de recharger l'entièreté du <svg>.

J'utiliserais aussi le fait de pouvoir utiliser le State interne du composant pour pouvoir stocker les données renvoyées par le backend, pour n'avoir à faire qu'une seule requête lors de l'affiche de la page.

Et surtout j'utiliserais la possibilité de faire de l'héritage entre composants pour éviter à avoir à créer des cas particuliers dans mon composant en fonction des différentes Map que je souhaite afficher. L'héritage me permettrais de mettre en commun les méthodes d'affichage qui sont utilisés pour les deux Map, tout en surchargeant les parties où les comportements sont différents.

Tout cela me fait dire qu'avec du recul une classe composant aurait été plus appropriée qu'un Hook, et que même si au début le Hook me paraissait opportun, au vu des différentes contraintes qui se sont ajoutés avec le développement du projet, mon choix de départ n'est plus du tout pertinent à la fin.

```
.on("mouseover", function(d) {
    let region_number = RegionEnum[d.target.__data__.properties.nom];
    if( location.pathname === '/'){
       RegionLastDataLoader(region_number).then( res => {
          covidData = res.data;
          setTextNameTooltip(`Région : ${d.target.__data__.properties.nom}`);

          if(covidData !== undefined){
             setTextDataTooltip(`Nombre de cas positifs : ${covidData.nbtest_positif}`);
          }
       })
    }
    else {
      DepartementLastDataLoader(d.target.__data__.properties.code).then( res => {
          covidData = res.data;
          setTextNameTooltip(`Département : ${d.target.__data__.properties.nom}`);

          if(covidData !== undefined && covidData !== null){
             setTextDataTooltip(`Nombre de cas positifs : ${covidData.nbtest_positif}`);
          }
      })
    }
```

Exemple typique non optimal/élégant : une requête à chaque *mouseover*, et une condition en fonction de quelle Map est affichée

Location

Le deuxième composant que j'ai réalisé est un composant qui va afficher un bouton « Localisez-moi » à l'utilisateur. Lors du clique sur ce bouton il est demandé à l'utilisateur s'il veut activer la Géolocalisation sur notre application, et s'il dit oui un point rouge va s'afficher sur la Map à ses coordonnées.

Pour ce composant je me suis encore une fois tourné vers les Hooks, car ils correspondaient bien au besoin que j'avais, à savoir : récupérer les coordonnées GPS de l'utilisateur et les transmettre à un autre composant. Pas besoin de state interne ni d'effets de bord, donc le hook était le candidat idéal.

A l'intérieur de ce composant je fais appel à l'API *Geolocation* de HTML5 qui va demander à l'utilisateur la permission et ensuite récupérer les données GPS.

J'ai rencontré un problème pour communiquer les données GPS avec le composant Map, car ils n'ont pas de lien de filiation direct (Ils ne sont pas parents-enfants).

Pour régler ce problème j'ai utilisé une des possibilités de React, *le Lifting State up.* Ce qui permet de faire remonter l'état courant d'un composant à son composant parent.

C'est de mon point de vue une manière de faire qui me semble élégante et optimale

```
const [localisation, setLocalisation] = useState([0,0])
const changeLocation = (coordinates) => {
  setLocalisation(coordinates);
}

return (
  <ThemeProvider theme={themeMode}>
    <div>
      <GlobalStyles />
      <div id="map">
        <h1 className="title">Bienvenue sur covid unfriendly</h1>
        <ToggleButtonTheme theme={theme} toggleTheme={toggleTheme} />
        <Router>
          <Switch>
          <Route exact path="/">
            <Map localisation = {localisation}/>
            <Location onChange={changeLocation}/>
          </Route>
```

Voici le composant parent qui contient mes deux composants Map et Location.

Ce que je fais c'est que j'utilise le hook d'état pour créer une variable que je peux changer grâce à une méthode déclarée dans ce composant parent.

Je donne en propriété à mon composant Location la méthode qui permet de mettre à jour cette variable.

```
function Location(props) {

    const geo = () => {
        if(navigator.geolocation){
            navigator.geolocation.getCurrentPosition((position) => {
             props.onChange([position.coords.latitude, position.coords.longitude]);
            });
        } else {
            alert("La géolocalisation n'est pas supportée par le navigateur")
        }
     }

    return(
        <Router>
            <button className="location" onClick={geo}>Localisez-moi</button>
        </Router>
    );
}
```

Lorsque je récupère les coordonnées j'appelle la fonction du composant parent.

Le composant parent met à jour la variable qui est envoyée au composant Map

```
function GeoChart({localisation}){

    //#region ···

    useEffect(() => {···
    } , [history,data,location,localisation]);
```

Et dans le composant Map je mets la variable localisation dans le tableau de dépendances du Hook d'effet. Ça aura pour effet de mettre à jour le render à chaque fois que la valeur change, donc de faire un render pour afficher le point rouge sur la Map.

Temps de développement par tâche

Map : 18h (beaucoup de temps passé dessus dû aux nombreux changements d'architecture que j'ai fais) (côté front)

Location : 3h (côté front)