28/02/2021

Rapport Projet web

Jrad Yassine

Yassine Jrad
FRONT-END

Identifiant github

JYassine

Tâches effectuées

Lors de ce projet j'ai réalisé les tâches suivantes :

- Réalisation des filtres (âge, région (sans la détection de géolocalisation), sexe, mois, année)
- Affichage des données en graphe
- Gestion des requêtes (gestion erreurs, loading : spinners) pour les filtres, affichage des données en graphe et affichage en table
- Gestion du thème (dark et light)
- Gestion du responsive pour le graphe et filtre

Stratégie pour la gestion des versions sur git

On a utilisé différentes branches pour travailler sur le projet, chaque branche correspond à une tâche du projet, lorsque la tâche est terminée, on faisait une pull request en attendant qu'un membre approuve pour merge ensuite sur une branche commune.

Solutions choisies

Axios + react-query

J'ai choisi d'utiliser Axios pour gérer les requêtes, j'ai principalement choisi cette librairie car axios gère « mieux » les erreurs que « fetch », voici un exemple très simple quand on execute ce code avec fetch :

Fetch(url).catch(error).then(….)

Avec fetch le then s'execute même s'il y a une erreur ce qui n'est pas le cas d'axios, à long terme cela peut entrainer des erreurs et créer des problèmes dans le code car ce n'est pas très intuitif. De plus axios transforme automatiquement la réponse en json.

J’ai choisi d’utilisé react-query pour faciliter la gestion des erreurs, le loading, ainsi qu’une meilleure lisibilité du code

Voici un exemple très simple de gestion d’erreurs avec axios

```
axios.get('/user/12345')
  .catch(function (error) {
    if (error.response) {
      // The request was made and the server responded with a status code
      // that falls out of the range of 2xx
      console.log(error.response.data);
      console.log(error.response.status);
      console.log(error.response.headers);
    } else if (error.request) {
      // The request was made but no response was received
      // `error.request` is an instance of XMLHttpRequest in the browser and an instance of
      // http.ClientRequest in node.js
      console.log(error.request);
    } else {
      // Something happened in setting up the request that triggered an Error
      console.log('Error', error.message);
    }
    console.log(error.config);
  });
```

On peut voir que le code peut très vite devenir inlisible si on a plusieurs requêtes. Mais ce n’est pas la seule raison, le code pour récupérer des données est répétée plusieurs fois dans le projet, il aurait fallu utiliser le hook « useState » pour gérer les trois états (error, loading, data) ainsi que « useEffect » pour récupérer les données avant le rendu, reactQuery fournit un hook « useQuery » qui permet de faire tout ceci en une ligne de code. C’est pour ces raisons que j’ai choisi d’utiliser cette librairie.

Recharts

Afin de créer le graphe j’ai choisi recharts comme librairie, j’ai choisi cette librairie car la documentation est claire, il y a plein d’exemples permettant de prendre en charge très rapidement la librairie, ce qui est important pour gagner du temps de développement. « React-chartjs-2 » est aussi une alternative présentant les caractéristiques citées, cependant mon choix s’est porté sur recharts car la librairie présente des composants responsives, ce que react-chart-js ne propose pas et qu’il faut gérer soit même.

Reactstrap

Nous avons choisi d’utiliser reactstrap pour gérer le responsive et le design des différents composants, Reactstrap propose un système de grille responsive où l’on peut placer nos composants (inspiré de bootstrap), une alternative à cette librairie est material-ui, qui propose un système de grille similaire, material-ui propose également un plus large choix de composants, notamment des boutons de design mobile. Cependant le choix proposé par reactstrap suffisait pour le projet, je n’ai pas vu de différence notable entre les deux, donc on a choisi de partir sur reactstrap.

Difficultés rencontrées

J'ai rencontré des difficultés dans la gestion du responsive sur la partie graphe, la librairie recharts propose un composant responsive pour les graphes mais sur mobile il ne s'adapte pas correctement, je n'ai pas eu le temps de résoudre ce problème, et je ne sais pas d'où le problème peut provenir malgré les tentatives.

Je n'ai pas eu de blocage sur les autres fonctionnalités. Les TP réalisés auparavant m'ont permis de comprendre les principes de React et d'être à l'aise avant le projet mais c'est surtout le choix des librairies qui m'a permis de ne pas avoir de blocages, chaque librairie me faisait gagner beaucoup de temps et masquer la complexité de certaines tâches où j'aurais eu des difficultés (comme le rendu 2D ou la gestion des états https (loading, error)).

Temps de développement / tâches

Pour le temps de développement il y a certaines tâches où j'ai mis plus de temps à réaliser que d'autres, voici approximativement le temps de développement pour chaque tâche.

Gestion du thème (dark et light) : 1h00

Gestion des requêtes pour les filtres (error et loading) : 2h00

Afficher les données en graphe : 4h00 (test des librairies recharts et react-chartjs-2)

Gestion du responsive : 3h00

Réalisation des filtres : 3h00

Le temps total est environ de 13h00, cela représente uniquement le temps de développement front-end.

Code

Solution élégante

```
function MonthsSelect(props) {
  const months = props.months;
  const listMonths = months.map((month) => {
    return (
      <DropdownItem
        key={months.indexOf(month) + 1}
        onClick={handleMonthFilter}
        value={month}
      >
        {month === "" ? "All Month" : month}
      </DropdownItem>
    );
  });
  return <DropdownMenu>{listMonths}</DropdownMenu>;
}
```

```
<Col>
  <h2>Mois</h2>
  <Dropdown isOpen={dropdownOpenMonth} toggle={toggleMonth}>
    <DropdownToggle caret> {monthFilter}</DropdownToggle>
    <MonthsSelect months={months} />
  </Dropdown>
</Col>
```

Je suis assez fier de ce code, c'est une fonction qui permet de créer un « DropDownMenu » avec les mois associés que je fais passer par les props. J'utilise la fonction map qui va créer un composant « DropDownItem » pour chaque mois. L'avantage est que c'est un composant réutilisable, et qui est dynamique (va se mettre à jour automatiquement s'il y a un changement dans la liste de mois)

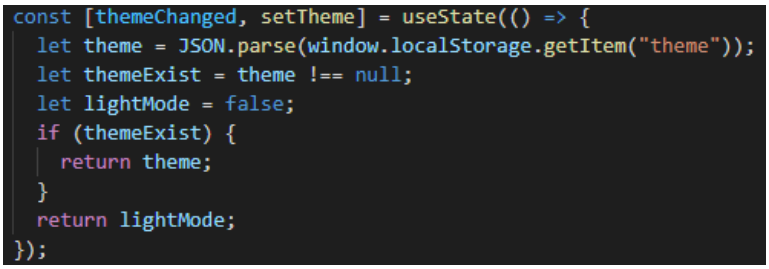
Solution à optimiser

Gestion des hooks : filtre et theme

```
useEffect(() => {
  const updateData = async () => {
    await monthQuery.refetch();
    await regionsQuery.refetch();
    await dataCovidQuery.refetch();
  };

  updateData();
  if (!monthQuery.isLoading && !monthQuery.isError) {
    setMonths(monthQuery.data);
  }
  if (!regionsQuery.isLoading && !regionsQuery.isError) {
    setRegions(regionsQuery.data);
  }
  if (!dataCovidQuery.isLoading && !dataCovidQuery.isError) {
    onChange(dataCovidQuery.data);
  }

  changeData(
    !monthQuery.isError ||
      !regionsQuery.isError ||
      !monthQuery.isLoading ||
      !regionsQuery.isLoading ||
      !dataCovidQuery.isError ||
      !dataCovidQuery.isLoading
  );
  onLoadingData(
    !regionsQuery.isLoading ||
      !monthQuery.isLoading ||
      !dataCovidQuery.isLoading
  );
  onError(
    dataCovidQuery.isError || regionsQuery.isError || monthQuery.isError
  );
}, [
  genderFilter,
  monthFilter,
  ageFilter,
  yearFilter,
```



```
const [themeChanged, setTheme] = useState(() => {
  let theme = JSON.parse(window.localStorage.getItem("theme"));
  let themeExist = theme !== null;
  let lightMode = false;
  if (themeExist) {
    return theme;
  }
  return lightMode;
});
```

Ici les deux solutions que j'ai réalisées peuvent être optimisées. On pourrait créer des hooks personnalisés qui permettrait que le code puisse être réutilisable. De plus si un autre développeur est amené à lire mon code, cela pourrait être assez fastidieux de devoir lire toute la logique derrière ce code, il serait mieux d'utiliser un hook personnalisé et de le nommer correctement pour comprendre directement son utilité.

Avec le thème on pourrait écrire useLocalTheme(theme), et dans le hook qui permet de mettre à jour les données (lors du changement de filtre) on pourrait créer un hook useUpdateData(listOfData) qui permettrait de mettre à jour les données dans la liste