Rendu projet Web

Maël Vaillant--Beuchot

Identifiant github : MaelVB

Tâches effectuées et temps de développement

Côté front :

Nom de la tâche	**Temps estimé**
Implémentation du thème de base de MaterialUI	30min
Développement du composant *AppDrawer* qui regroupe une barre de navigation (*AppBar*) et un menu vertical (*Drawer*)	1h
Développement du contenu des pages List, Graph et Map avec l'aide de Maël Delaby	5h
Développement du bouton de la page Map qui, à l'aide de la géolocalisation, renvoie une information concernant le département de l'utilisateur	1h00
Développement de la gestion du multi-thème pour intégrer le *switch* de thème	3h
Développement du *switch* pour passer d'un thème sombre à un thème clair et intégration de la fonction *prefers-color-scheme* qui met automatiquement le site en *dark mode* si l'utilisateur n'a pas encore utilisé le *switch* et que son OS a le *dark mode* en préférence	1h30

Côté back :

Nom de la tâche	**Temps estimé**
Développement de la récupération des CSV et de leur insertion dans notre base de données	5h
Développement de la récupération des données utilisées dans les pages List, Graph et Map avec Maël Delaby	3h

Développement d'une première version des schémas du modèle de données	1h

Stratégie employée pour la gestion des versions avec Git

Au niveau de GitHub :

- Les différentes tâches ont été créées sous forme d'*issues* en début de projet
- Chacun devait s'assigner la tâche sur laquelle il travaillait et la déplacer dans le kanban

Au niveau des branches Git :

- Une *issue* est égale à une branche, quand la personne a fini de travailler sur son issue, elle fait une *pull request* qu'une autre personne doit accepter pour qu'elle soit *merger* dans la branche 'dev'
- Lorsque plusieurs branches ont été *merge* dans 'dev' et que cette dernière a été testée, elle est *merge* dans la branche 'master'

Solutions choisies

Le premier des choix techniques qui a été fait est celui du *framework*, nous avons choisi MaterialUI car déjà connu de plusieurs membres du projet et correspondant aux différentes contraintes de visualisations présentes dans le projet : représentation sous forme de tableau et de graphe notamment. D'autres *framework* auraient pu être utilisés comme React Bootstrap ou Ant Design.

Concernant mes tâches, j'ai dû faire plusieurs choix. D'abord pour la navigation entre les pages, le routage a été effectué par Othmane, mais pour la partie visuelle j'avais plusieurs choix de menu et de présentation des informations (des composants *Menu*, des *Accordions*, une *AppBar*, un *Drawer* etc). J'ai opté pour l'*AppBar* couplé au *Drawer* car ce sont des composants dont le style est déjà responsive avec le thème par défaut de MaterialUI, nous n'avons donc pas à retravailler la CSS du thème, ce qui fait gagner du temps. Ensuite, concernant les différentes pages, le tableau est celui de MaterialUI qui dispose d'ors et déjà de fonctions de tri, pour la page du graphe nous avons utilisé *recharts* une bibliothèque recommandée par MaterialUI et donc s'intégrant parfaitement avec les autres composants du *framework*, un autre package *npm* aurait également pu fonctionner comme *react-chartjs-2*. Concernant la page Map, nous avons hésité à utiliser une vraie map venant de Google Maps ou MapBox, nous avons opté pour une map dessinée sous forme d'une image fixe car bien plus rapide à intégrer à mon sens, grâce au package *@socialgouv/react-departements*. Le revers est que nous ne pouvons colorier les zones qu'en deux couleurs, c'est pour cela que nous n'avons qu'un seul palier du taux d'incidence dans la légende.

Concernant la récupération de la zone géographique de l'utilisateur, plusieurs solutions étaient possibles : une API de cartographie comme Google Maps ou une base de données renseignant des informations sur la zone géographique à partir d'un point défini par sa latitude et sa longitude. C'est cette dernière solution que j'ai retenu car, toujours dans une problématique de gagner du temps, les appels à des API de cartographie requiert toute la création d'un compte et l'utilisation d'une clé d'API, alors que le site que j'ai trouvé

(https://geo.api.gouv.fr/communes), permet de faire des appels d'API en toute simplicité, sans avoir besoin d'une clé, ainsi, après récupération de la latitude et de la longitude de l'utilisateur, je peux récupérer son département grâce à un appel à une API du gouvernement.

Pour le changement de thème j'ai opté pour un simple bouton de style *switch* car plus rapide à mettre en place qu'une modale, le revers de cette solution est que je n'avais pas de modale sous la main lorsque j'ai développé la reconnaissance du thème préféré de l'OS, il n'y a donc aucune information émanant de l'interface lorsque le site reconnaît que l'utilisateur préfère le mode *dark* (c'est une simple entrée dans le Local Storage qui est effectuée). De plus, un problème de rafraichissement des composants sur lequel nous n'avons pas eu le temps de nous pencher fait que le changement de thème est visible une fois, mais pas au deuxième test, il faut d'abord rafraîchir la page pour que le changement soit à nouveau visible.

Difficultés rencontrées

La difficulté de la création de la page Map est intervenue au moment de choisir la technologie, comme je n'avais jamais utilisé une API de cartographie j'avais peur de passer trop de temps dessus et même après quelques recherches et donc une certaine perte de temps, j'ai préféré trouver une alternative plutôt que de passer plus de temps dessus.

Une autre difficulté a été au niveau de la récupération et de l'affichage des données dans les pages List et Graph. En effet, étant un domaine avec des termes particuliers et beaucoup de données, j'ai mis du temps à correctement utiliser ces données, en prenant des paramètres qui sont vraiment intéressants. La solution a tout simplement été d'aller se renseigner en profondeur sur le site du gouvernement d'où sont tirées les données pour pouvoir afficher les informations qui nous intéressent.

Enfin, une dernière difficultée a été au niveau du changement de thème, voulant un développement rapide, je suis parti sur un simple *switch* alors qu'une modale aurait été plus pertinente notamment pour l'affichage du message lorsque l'application a détectée la préférence de l'OS de l'utilisateur.

Code

Un composant que je trouve assez optimal est celui qui affiche la Map, je pense que la difficulté d'utiliser une API de cartographie a été correctement contournée. Pendant que je récupère les données qui me permettent de colorer les département en fonction d'un seuil, j'affiche un *loader*. Une fois que les informations sont récupérées, je test toutes celles qui sont au-dessus du seuil et je les mets dans la variable qui sera passée au composant de la bibliothèque qui affiche la carte. Enfin, j'affiche également un bouton qui permet à l'utilisateur de connaître la valeur précise du département dans lequel il se trouve grâce à la géolocalisation et une API du gouvernement. Le tout est assez rapide.

```
…
// Récupération des données
const fetchData = async () => {
const result = await axios.get(getCovidDataUrl);
setDepData(result.data);
let selectedDep = [];
for(let i=0; i<result.data.length; i++){
if(result.data[i].P > critP) {
```

```
selectedDep.push(result.data[i].dep)
}
}
setDepSelected(selectedDep);
setLoading(false);
};

useEffect(() => {
fetchData();
}, []);
```

// Récupération de la géolocalisation de l'utilisateur

```
const getLocation = async () => {
if (navigator.geolocation) {
await navigator.geolocation.getCurrentPosition(showPosition);
}
}
```

// Récupération du nom du département à partir de la géolocalisation

```
const showPosition = async (position) => {
console.log("Get location")
const departmentUrl = "https://geo.api.gouv.fr/communes?lat=" + position.coords.latitude + "&lon=" + position.coords.longitude + "&fields=departement&format=json&geometry=centre"
const departmentInfo = await axios.get(departmentUrl)

let line = '';
let i=0;
while (i<depData.length && line == '') {
console.log(i)
if(depData[i].dep === parseInt(departmentInfo.data[0].departement.code)) {
line = "Votre département : " + departmentInfo.data[0].departement.nom + ' (' + departmentInfo.data[0].departement.code + ') a pour taux d\'incidence P : ' + depData[i].P;
}
i++;
}
setLocationData(line)
}

return (
<>
<Typography>
Rouge : Incidence supérieure à {critP}
</Typography>
{ loading ? <CircularProgress /> :
<>
<France color="#556cd6" highlightColor="#c80000" departements={depSelected} />
<Typography>
Cliquez sur le bouton ci-dessous pour accéder à l'incidence de son département :
</Typography>
<Typography>
{ locationData === '' ? <Button onClick={getLocation}>Récupérer l'information pour mon département</Button> : locationData }
</Typography>
</>
}
</>

)
```

}

Le composant que j'aimerais refaire est celui du *switch* de thème pour avoir quelque chose plus proche du style de Twitter avec une modale permettant de sélectionner plusieurs thèmes et d'avoir la possibilité d'afficher du texte concernant les thèmes, avec par exemple une sélection automatique en fonction du thème de l'OS de l'utilisateur. Cette refactorisation permettrait par la même occasion trouver pourquoi ce changement de thème ne rafraîchit les autres composants qu'une seule fois et pas la deuxième fois.

```
export default function ThemeModal() {
  const { currentTheme, setTheme } = useContext(CustomThemeContext)
  const isDark = Boolean(currentTheme === 'dark')

  const handleThemeChange = (event) => {
        const { checked } = event.target
        console.log(event.target)
        if (checked) {
        setTheme('dark')
        localStorage.setItem('appTheme', 'dark')
        } else {
        setTheme('normal')
        localStorage.setItem('appTheme', 'normal')
        }
        }

  const osThemePreference = () => {
        const userPrefersDark = window.matchMedia &&
window.matchMedia('(prefers-color-scheme: dark)').matches;

        if(userPrefersDark && (localStorage.getItem('appTheme') == undefined)){
        localStorage.setItem('appTheme', 'dark')
        }
  }

  useEffect(() => {
        osThemePreference();
  }, []);

  return(
        <FormControlLabel
        control={<SwitchUI checked={isDark} onChange={handleThemeChange} />}
        label="Change theme"
        />
        )
}
```

A la place d'un composant Switch, je mettrais un bouton qui ouvre une modale (composant Modal avec MaterialUI) avec dedans un groupe de bouton (ButtonGroup) permettant de sélectionner un thème et un message indiquant que le bouton du *dark* thème a été coché par défaut à partir de l'information venant de l'OS de l'utilisateur et que l'utilisateur peut décocher cette fonctionnalité via une simple case à cocher.