Livrable Programmation Web Client-Side

Graulier Brice GraulierBrice (github.com)

Tâches effectuées :

La majeure partie de mon travail sur le front-end fut la mise en place des différents thèmes (le projet en à trois : light, dark et plague). Avec le template sur lequel nous nous somme basé il n'a pas été trop difficile de mettre en place différents thèmes. J'ai tout simplement créé des variables css qui vont contrôler la couleur de l'affichage. Il y a eu un travail pour comprendre l'architecture du template mais une fois familiarisé avec l'arborescence des fichiers le travail était rapide.

La partie plus technique de ce travail était la mise en place du *local storage* pour garder en mémoire le thème sélectionné pour que celui-ci réapparaisse même si la page est rechargée. La variable *theme* du l*ocal storage* garde en mémoire un string qui détermine le thème actuel. Ce string est utilisé pour référencer les variables css à utiliser pour l'affichage.

Un dernier détail des thèmes est la détection des préférences du navigateur. Cette fonctionnalité était simple à mettre en place car un simple *matchMedia* permet de détecter facilement les préférences du navigateur.

Pour la partie back j'ai exposé plusieurs routes pour mettre à dispositions des données pour le front. J'ai fait des routes pour donner des recherches sur la date et le sexe des cibles. Il a fallu un peu de traitement pour avoir le tri par année et par mois car la variable jour des données inclue année-mois-jour. Une expression régulière est utilisée pour match les bonnes données. J'ai aussi rajouté des requêtes dans la collection Postman pour aider l'équipe à bien comprendre comment utiliser les routes.

Stratégie employée :

Pour les thèmes j'ai décidé de passer par des variables css. Ce choix vient principalement du fait que le template que nous utilisons passe par cette méthode pour faire le style de l'application. C'était tout à fait raisonnable pour ce que nous avions à mettre en place car le style de notre page est global et ne varie pas entre les différents éléments.

L'alternative aurait été d'utiliser des *themed components* pour styliser tous les components mais ça aurait demander un lourd refactor et l'avantage de pouvoir avoir un style différent sur les components ne nous servait pas, je ne voyais donc pas de raison de faire de cette manière.

Difficultés rencontrées :

Je n'ai pas rencontré de difficultés bloquantes juste quelques soucis liés à l'apprentissage de React. J'avais du mal à transmettre directement des *props* à travers mes *Route* j'ai donc dû passer des fonctions qui renvoyais mon component en attribut de la Route. Un autre petit blocage était pour que mon component toggle change le mode des

thèmes. J'ai vite trouvé qu'il fallait faire passer une fonction de mon App.js jusque dans le component voulu pour que celui-ci puisse effectuer le *setStorageMode.*

Temps de développement :

Front :

Découvrir le template : 1h
Themes CSS : 4h30min
Local Storage : 1h
Détection du dark mode : 30min

Back :

Routes date : 45min
Route sexe : 15min
Collections Postman : 15min

Rédaction Git (wiki et readme) : 2h30min

Code :

Une belle fonction :

```
    const handleChangeMode = useCallback(
        (mode) => {
            setStorageMode(mode);
            document.documentElement.style.setProperty('--current-gradient', `var(-
-${mode}-gradient)`)
            document.documentElement.style.setProperty('--bg-current',`var(--bg-
custom-${mode})`)
        },
        [setStorageMode],
    );
```

Je trouve cette fonction assez succincte et bien compréhensible. En simplement 3 lignes toute l'affichage de la page est changé. Les variables sont claires et il est facile de comprendre son fonctionnement de l'enrichir si quelqu'un venait à travailler sur le projet.

Une fonction moins belle :

```
    public async getHospitalDays(req: Request, res: Response) {
        if(req.params.year){
            if(req.params.month){
                if(req.params.day)
                    return res.status(200).json(await HospitalDayModel.find({jour:
{$regex: "^"+req.params.year+"-"+req.params.month+"-"+req.params.day}}));
                return res.status(200).json(await HospitalDayModel.find({jour:
{$regex: "^"+req.params.year+"-"+req.params.month}}));
```

```
            } return res.status(200).json(await HospitalDayModel.find({jour:
{$regex: "^"+req.params.year}}));
        } return res.status(200).json(await HospitalDayModel.find());
    }
```

Le problème de cette fonction est vite vu. Dans l'effort d'écrire moins de fonctions j'ai quatre routes qui font appel à la même fonction. Le résultat est fouillis et dure à comprendre. Il faudrait faire quatre nouvelles fonctions, une par route, pour plus de netteté et lisibilité. De plus, cette cascade de if est inefficace et retarde le temps de réponse.