Damien Piedanna

Programmable web (client-side)

Rapport individuel

Identifiant Github

damien-piedanna

Tâches effectuées

Au début du projet, nous avons fait le choix dans notre équipe d'avoir 2 personnes responsables de la partie backend et les 2 autres personnes sur la partie frontend. J'ai fait partie de l'équipe backend.

Du côté du frontend, je me suis concentré sur l'authentification d'un utilisateur, celui-ci a la possibilité de renseigner ses pleins d'essences au cours du temps et ainsi avoir des statistiques sur sa consommation. Nous voulions aussi donner la possibilité à nos utilisateurs de changer le thème de l'application (clair ou sombre), ce choix est persistant.

Côté backend, j'ai travaillé en collaboration avec Nicolas et Nathan pour tout ce qui concerne la récupération des données sur les stations (tries, filtres, colonnes de retours, …).

Stratégie employée pour la gestion des versions avec Git

À l'initialisation du projet nous avons utilisé le système d'issues de Github pour nous répartir les tâches de façon équitable et avoir un aperçu global des attendues, notamment pour définir le payload entre le back et le front. Nous référençons ses issues au moment de faire un commit. Une branche est créée pour les grosses features puis merge dans « develop » une fois complété. La branche « develop » est merge sur « main » au moment où nous voulons mettre à jour la production (déploiement automatique sur le cloud).

Solutions choisies

Pour notre projet, nous avons fait le choix d'utiliser le fremework « mui » qui nous mettait à disposition des composants préfabriqué et customisable car l'un des membres avait déjà une expérience dessus.

Pour que les utilisateurs puissent sauvegarder le thème qu'ils choisissent, nous avons utilisé le localStorage car c'est une solution simple à utiliser, rapide à s'exécuter et contrairement au cookie, cette solution stocke les informations dans le navigateur de client et non sur nos serveurs, cela tombe bien car cela ne nous serait pas utile.

Pour sauvegarder l'identifiant de connexion, nous avons fait le d'utiliser Redux car nous avions besoin de stocker dans le store un booléen pour savoir si l'utilisateur est connecté et un token de connexion consultable un peu de partout dans notre application.

Difficultés rencontrées

La principale difficulté que j'ai rencontrée et la familiarisation avec tous les concepts spécifiques à React et le fait d'arriver plus tardivement dans le projet, étant principalement concentré sur la partie backend.

Temps de développement / tâche

Une demi-journée pour l'implémentation du thème, une autre demi-journée pour la connexion.

Code

Ce code est déclenché lorsque l'utilisateur envoie le formulaire de connexion.

Premièrement, on vérifie si les states « username » et « password » sont renseigné par l'utilisateur (ces states sont modifié automatiquement lorsque l'utilisateur modifie le champ). On vérifie ensuite si on est dans un mode « se connecter », sinon on sera dans un mode « s'enregistrer » (dans notre application pour des raisons de simplicités, vous avez la possibilité de créer un compte sans adresse mail).

Si l'utilisateur est en mode « s'enregistrer » on va alors faire un appel à notre api avec le nom d'utilisateur et le mot de passe afin de créer le compte, si tout se passe bien on connecte alors l'utilisateur automatiquement en faisant un appel récursif, sinon on catch les erreurs via la fonction handleError() va afficher au-dessus du formulaire le message d'erreur, typiquement cela peut être « Nom d'utilisateur déjà utilisé ».

En mode « connexion », nous allons enregistrer via Redux le fait que l'utilisateur est connecté ainsi que son token qui sera ensuite utilisé pour les requêtes liées aux historiques de pleins.

```
public handleSubmit() {
    if (this.state.username && this.state.password) {
        if (this.state.signIn) {
            this.authApi.signIn(this.state.username, this.state.password).subscribe((res)=>{
                if(res){
                    this.props.dispatch(updateIsLogged(true));
                    this.props.dispatch(updateToken(res.data.access_token));
                    this.props.closeLoginPopup();
                }
            },err => this.handleError(err));
        } else {
            this.authApi.signUp(this.state.username, this.state.password).subscribe((res)=>{
                if(res){
                    this.switchSignInOruP();
                    this.handleSubmit();
                }
            },err => this.handleError(err));
        }
    }
}
```

Afin de récupérer le token de connexion dans nos services faisant appel à notre backend, j'ai créé une fonction qui permet de le facilement récupérer cette info, on profite ici du fait que Redux persiste son état dans le local storage pour aller le récupérer de façon brute, c'est une utilisation détournée qui n'est pas optimale mais qui a pour avantage d'être utilisé facilement dans votre service.

```
public static getToken() {
    const state = localStorage.getItem("persistantState");
    if (state) {
        return JSON.parse(state).userLogged.token;
    }
}
```