Programmable Web – Client Side: Rapport individuel
Lisa-Marie DEMMER

Identifiant github : Lisa-Demmer

Tâches effectuées

Au début du projet commun aux cours de Programmable Web Server Side et Client Side, nous avons fait le choix de diviser notre équipe de quatre en deux équipes de deux personnes, une équipe sur l'application côté Back, et l'autre sur l'application côté Front. J'ai fait le choix de faire partie de l'équipe assignée au développement de l'application côté Back.
Pour la partie Front, j'ai développé un composant qui permet de consulter les statistiques de plein de carburant d'un utilisateur. Avec notre application, un utilisateur peut renseigner ses pleins de carburants, puis en accédant à la page de statistiques des pleins, il peut voir, par carburant, le nombre de pleins effectués, le prix moyen au litre et également un graphique montrant les différents pleins effectués, leur prix au litre ainsi que leur date.
Côté back, j'ai également créé la route retournant les différentes informations requises par ce composant, route que j'ai pu appeler depuis le service en charge de faire les appels en rapport avec les pleins de carburants d'un utilisateur.

Gestion des versions avec Git

Pour gérer les versions avec Git, des branches ont été créées pour chaque fonctionnalité et nommées de cette manière : feature/#{issueNumber}-{issueShortDescription}. Des issues ont été créées pour chaque fonctionnalité et distribuées en fonction des disponibilités et des attirances de chacun.

Solutions choisies

Je n'ai pas eu à faire de réels choix de librairie, ou de choix de technique à utiliser, étant donné que je travaillais sur le côté Back, et que sur la partie Front, j'ai travaillé pour faire un seul composant afin de me faire la main sur React. La majorité de l'application était déjà terminée, ce sont les personnes de l'équipe dédiée à la partie Front qui ont fait ces choix, j'ai utilisé les mêmes principes qu'eux afin de garder une bonne homogénéité dans toute l'application.
Le seul choix que j'ai potentiellement eu à faire, c'est le choix de la librairie pour faire le graphique des différents pleins d'un utilisateur. Mais des graphiques étaient déjà présent dans l'application et la librairie utilisée était Recharts. J'ai vérifié que la librairie permettait de faire ce que je souhaitais, et étant le cas, je n'ai pas cherché à utiliser une autre librairie. En cherchant sur internet, cette librairie arrive en première place de nombreux classements sur les librairies permettant de faire des graphiques avec React, mais il existe bon nombre d'alternatives dont certaines étant utilisable sans React comme la librairie amcharts qui est très utilisées dans le monde Javascript.

Difficultés rencontrées

La principale difficulté rencontrée a été le fait d'arriver dans un projet React inconnu et de devoir s'approprier l'application afin de pouvoir l'étendre en plus d'apprendre le fonctionnement même de React. Ayant déjà utilisé des Framework avec un fonctionnement en composant, l'adaptation fût quand même assez rapide.

Temps de développement / tâche

Pour le composant que j'ai effectué sur la partie Front, cela m'a pris environ une demi-journée le temps de m'approprier React et le projet en lui-même.

Code

Je vais commenter les méthodes les plus importantes que j'ai implémentées dans mon composant et les critiquer.

```
/**
 * Load gas data for chart
 * @private
 */
private loadGasData(gas_name: string):void {
    const gasRefuel = this.getGasRefuelByGasName(gas_name);
    const gasData: any = [];
    if(gasRefuel) {
        for (const userGasRefuel of gasRefuel.list) {
            const date = new Date(userGasRefuel.date);
            gasData.push({
                date: date.toDateString(),
                price: (userGasRefuel.total_price / userGasRefuel.quantity).toFixed(2),
            })
        }
    }

    this.setState({
        gasData: [
            {gas: gas_name as GasType, data: gasData},
        ]
    });
}
```

Figure 1 : Code méthode loadGasData

Cette méthode est la méthode en charge de récupérer l'objet contenant les informations des pleins d'un type de carburant et, pour chacun des pleins effectués, rempli un tableau avec la date et le prix au litre de ce plein. Enfin, on stocke dans l'état du composant les données nécessaires pour le graphique.

Cette méthode n'est pas optimale, premièrement, cela pourrait être utile de gérer les problèmes de division par 0 si un utilisateur a réussi à renseigner un plein avec une quantité de 0. Ensuite, de faire un « cast » du nom du carburant en un enum va poser un problème le jour où un nouveau carburant est crée, mais que notre enum n'est pas modifié. Si la date du plein est nulle, nous aurons un plein avec une date invalide alors qu'il faudrait au moins ne pas l'afficher dans le graphique, et potentiellement affiché une alerte à côté du nombre de plein afin d'avertir l'utilisateur qu'il a des pleins avec des données non-valide. Ces vérifications doivent être faites au moment de renseigné un plein et également au niveau du Back afin de ne pas arriver à ce cas de figure. Pour finir, l'utilisation de cette méthode n'est pas optimale, mais je vais y revenir en parlant de la prochaine méthode.

```
/**
 * Load refuel stats
 * @param period
 * @private
 */
private loadRefuelStats(period: string):void{
    this.setState({ gasRefuelStats: [], gasList: [], selectedGas: '', gasData: [] });
    this.refuel_stats_request = this.refuelApi.getRefuelStats(PeriodEnum.ALL).subscribe((gasRefuelStats: GasRefuelStatsModel[]) => {
        const gasList = gasRefuelStats.map((gasRefuelStat: GasRefuelStatsModel) => {return gasRefuelStat.gas_name});
        this.setState({gasList: gasList});
        this.setState({gasRefuelStats: gasRefuelStats});
        this.setState({selectedGas: gasList[0]});
        this.loadGasData(gasList[0]);
    });
}
```

Figure 2 : Code méthode loadRefuelStats

Cette méthode permet de faire l'appel à l'api pour récupérer les informations des pleins de carburant d'un utilisateur, et remplit l'état du composant avec les données nécessaires. Cette méthode n'est également pas optimale, car le premier **SetState** n'est pas utile puisque l'état est déjà initialisé avec ces valeurs et qu'il faut limiter le nombre d'appels à cette méthode. Ensuite, il y a plusieurs appels consécutifs qui peuvent être rassemblés en un seul appel.

Pour finir, comme je l'ai dit précédemment, l'utilisation de la méthode **loadDasData** n'est pas optimale. En effet, celle-ci est appelée au début avec le premier carburant de la liste, puis à chaque changement de carburant. Il aurait été plus judicieux de faire une méthode qui calcule les données pour chaque carburant une seule fois et qui met ces données dans l'état du composant. Puis ensuite une méthode qui récupère dans l'état, les données du carburant sélectionné pour afficher le bon graphique. Une meilleure optimisation serait que le Back renvoie directement ces données afin que côté Front, il ne reste plus qu'à récupérer la bonne donnée dans l'état. Cette optimisation permettrait d'envoyer seulement les données indispensables pour le Front.