Ralph EL CHALFOUN Rapport Individuel Programmable Web Client

Identifiant GitHub :

iBananos : https://github.com/iBananos

Tâches effectuées :

Je me suis personnellement plus concentré sur la partie backend, cependant j'ai quand même contribué au frontend en réalisant les tâches et fonctionnalités suivantes :

- J'ai pris en charge le formulaire de la position et de l'essence demandée et son envoie au serveur afin qu'il retourne les stations proches de cette position fournissant cette essence.
- J'ai ajouté la possibilité de rechercher via deux critères, la station la plus proche ou la moins chère, (ceci change uniquement l'affichage du tableau réalisé par Ossama ASHRAF).
- J'ai ensuite récupéré les stations et calculé les distances avec l'utilisateur, j'ai créé les filtres en fonction des prix, horaires et services des stations récupérées.
- J'ai géré les filtres afin que lorsqu'on en sélectionne/désélectionne un, la Map se rafraichit et certaines stations apparaissent/disparaissent.
- J'ai coloré les stations en fonction du prix affiché pour l'essence demandé :
    - Le prix le plus bas sera vert
    - Le prix le plus haut sera rouge
    - Entre ces deux valeurs la couleur sera dans le dégradé de vert à rouge en fonction du prix.
- J'ai implémenté un système de connexion/inscription basique.
- J'ai déployé l'application sur heroku, pour cela j'ai du crée un repository GitHub spécial pour heroku, j'ai build le client dans les fichiers du serveur afin de ne déployer que les fichier serveur, j'ai dû ajouter un Procfile et modifier les adresses d'échange entre le serveur et le client. Ainsi l'application est disponible sur : https://acence.herokuapp.com/
- J'ai contribué avec mon groupe aux styles de certains component et au fait que l'application soit responsive.
- J'ai ajouté la possibilité de remplir l'adresse directement via la position GPS en récupérant la latitude et longitude de l'utilisateur et en trouvant une adresse proche de ces donnée grâce à l'api open.streetmap.
- Sur le serveur j'ai en fonction des requêtes ajouté des messages aux réponses pour pouvoir traiter différents cas sur le client :
    - Par exemple pour l'inscription si l'adresse mail est déjà utilisée le serveur renvoie un message qui est récupéré sur le client qui l'affiche ensuite.

(Entre autres sur le serveur j'ai créé les routes pour les requêtes du client à l'exception des favoris. J'ai réalisé l'intégralité des tests sur le serveur. J'ai requêté la base de données pour renvoyer au client les stations qui ont une position proche de celle demandée. J'ai vérifié tous les éléments du body des requêtes du client pour éviter des bug ou attaques pouvant « faire planter » le serveur.)

Temps de développement / tâche :

Les systèmes de connexion et inscription ne m'ont pas pris beaucoup de temps, une après midi, cependant si l'on voulait rendre ce système plus sécurisé il faudrait passer plus de temps à créer des tokens et les stocker dans les cookies et localStorage.

Les filtres ont été la tâche la plus longue à implémenté, plusieurs jours, gérer les différents filtres services, ouverture, prix et en fonction de ces derniers: le réaffichage des stations.

Les différentes requêtes avec le serveurs furent rapides à mettre en place grâce à une méthode que j'expliquerai par la suite.

La coloration des points sur la map m'a pris quelques heures pour créer une fonction qui colore correctement chaque station.

Pour le reste des tâches c'est de l'ordre de l'heure/demi-heure.

Diffiultés :

Pour la partie Front-End je n'ai pas rencontré de difficulté particulière hormis le deployement sur heroku qui a nécessité quelques recherches et modifications (mais c'est plus du back-end).

Fonction qui me semble optimale :

```
static async sendRequest(methode , url , data , callback) {
        document.getElementById("loading").style.visibility = "visible";
        var xhr = new XMLHttpRequest()
        console.log(hostname+url)
        xhr.open(methode, hostname+url, true);
        xhr.setRequestHeader("Content-Type", "application/json");

        xhr.addEventListener('readystatechange', function(e) {
            if (xhr.readyState === 4 && xhr.status === 200) {
                console.log(JSON.parse(this.response))
                if(JSON.parse(this.response).status ==="500"){
                    alert(JSON.parse(this.response).message)
                }else{
                    callback(this.response);
                }
            }
            else if (xhr.readyState === 4 && xhr.status !== 200) {
                alert("Une erreur est survenue. Veuillez rafraîchir la page");
            }
            document.getElementById("loading").style.visibility = "hidden";
        });
        xhr.send(data);
    }
```

Cette fonction prend en argument :

- methode : ‘POST’ ou ‘GET’
- url : la route vers le serveur sans son adresse exemple ‘/login’ ‘/addFavorite’
- data : les données à ajouter au corps de la requête ex un JSON
- callback : une fonction à appeler après avoir reçu la réponse du serveur

Cette fonction est disponible en appelant la classe Utils depuis n’importe quel component/page, une fois la fonction appelée un écran de chargement apparait (réalisé par Elise CHAMBERLIN), et une requête XMLHttpRequest est envoyée en fonction des arguments.

Une fois la réponse reçue : si dans le corps de la réponse nous avons un status différent de 200 alors c’est qu’une erreur s’est produite sur le serveur, nous alertons donc le client de l’erreur en affichant le message associé. Si le status est 200 alors le serveur considère que tout s’est passé correctement et on appel le callback passé en argument. Et on enlève l’écran de chargement.

Je trouve cette fonction optimale car elle facilite les échanges entre le client et le serveur et permet de communiquer simplement et de façon régulière. Si nous avons besoin d’ajouter une route il nous suffit de changer le paramètre « url » et d’appeler n’importe quelle nouvelle fonction en callback.

De plus lors du déploiement sur heroku je n’ai eu besoin de modifier qu’une fois le hostname qui est déclaré plus haut dans la classe, et non dans tous les fichiers du client.

Fonction qui aurait besoin d’être améliorée :

```
createSettings(response) //crée les filtres à gauche de la map
createListPoint()  //détermine en fonction des filtres les stations à afficher
```

Ce ne sont pas ces fonctions en particulier mais le système pour créer les filtres à côté de la map (situé sur la page Home). A l’heure actuelle nous devons parcourir toutes les stations et leurs services pour déterminer les checkbox services et pour chaque service : combien de stations le propose.

De plus nous devons vérifier les ouvertures et fermetures pour la checkbox (ouverte), et récupérer le prix le moins chère et le plus chère pour l’input range du prix.

A chaque changement d’état des checkbox ou de l’input range, nous reparcourrons les stations pour savoir lesquels doivent disparaitre ou apparaitre ce qui fait beaucoup de calcul. Dans le cas d’un grand nombre de station pour une localisation, les calculs peuvent prendre plus d’une seconde, là où nous souhaitions que ce soit presque instantané. Ces calculs étant fait par le client, selon la puissance de l’appareil ce temps varie.

Une stratégie pour rendre cela plus optimal serait de savoir directement quelles stations retirer de la map lors d’un changement. Ou bien faire certains calculs (distance/statut d’ouverture) en amont par le serveur. De plus les filtres pourraient faire partie d’un component et être sortis de la page Home.