Ossama ASHRAF

Rapport Individuel Programmable Web Client

Identifiant GitHub :

Ossama98 (https://github.com/Ossama98)

Tâches effectuées :

Personnellement j'ai travaillé sur les deux parties du projet ; la partie backend ainsi que la partie frontend. Concernant la partie frontend, j'ai implémenté les tâches et fonctionnalités suivantes :

- J'ai construit un tableau qui permet d'afficher les résultats des recherches des utilisateurs de notre site web. Il est trié par ordre croissant des prix ou des distances selon le type de recherche effectué (recherche des stations moins chères ou plus proches respectivement).

  Ce tableau contient les informations suivantes :

  - Les stations trouvées
  - Le prix de l'essence recherché pour chaque station
  - La distance qui sépare l'utilisateur de chaque station d'essence
  - Pour chaque station une indication pour savoir si la station est ouverte ou fermé au moment de la recherche
  - Une étoile qui permet d'ajouter/retirer une station des favoris
  - Pour chaque station un bouton "plus d'infos" que je vais détailler par la suite
- J'ai implémenté un canvas qui apparaît à la suite d'un clic sur le bouton « plus d'infos ». Ce dernier contient :
  - L'adresse de la station essence
  - Les services proposés par la station d'essence
  - La présence ou non d'un automate de paiement 24h/24
  - Les horaires de la station d'essence
  - Les prix des essences disponibles dans la station d'essence
  - Un graph pour visualiser l'évolution des prix des essences disponibles dans la station d'essence
- J'ai mis sur la carte toutes les stations d'essence trouvées ainsi qu'un popup qui apparaît si on clique sur n'importe quelle station. Ce popup contient :
  - L'adresse de la station essence
  - Les essences disponibles dans cette station d'essence

        - Un bouton pour voir le chemin de la position actuelle de l'utilisateur vers la station d'essence

    J'ai décidé de ne pas mettre plus d'informations dans ce popup pour pas qu'ils soient trop chargés. Pour plus d'infos il faudra cliquer sur le bouton "plus d'infos" dans le tableau.

- J'ai contribué avec mon groupe aux styles de certains component et au fait que l'application soit responsive.
- J'ai implémenté une page 404 Not Found. Cette dernière est affichée lorsque l'utilisateur essaie d'accéder à une page qui n'existe pas.
- J'ai mis en place un mode jour/nuit. Cela donne à l'utilisateur la possibilité de changer le thème de l'application. Le choix de l'utilisateur est sauvegardé ainsi si la page est rechargée le thème reste le même.

(Entre autres sur le serveur j'ai mis en place la connexion avec la base de données MongoDB. J'ai automatisé la mise à jour des données des stations d'essence sur MongoDB ainsi que la conversion de ces données du format XML en format JSON ce qui constitue une étape nécessaire pour pouvoir importer ces données sur MongoDB).

Stratégie employée pour la gestion des versions avec Git :
Nous avons créé des issues et nous nous sommes attribués ces derniers pour pouvoir avancer sur des tâches différentes en même temps.

Temps de développement / tâche:

- La page 404 Not Found ne m'a pas pris beaucoup de temps à mettre en place.
- Le mode jour/nuit m'a pris une après-midi pour mettre en place. Cependant j'ai mis du temps supplémentaire pour adapter toutes les pages au mode dark(nuit).
- Le tableau et le canvas m'ont pris quelques heures pour mettre en place. Cependant je les ai souvent modifiés pour changer leur style et/ou les informations qu'ils contiennent etc.
- Pour le reste des tâches c'est de l'ordre de l'heure/demi-heure.

Difficultés rencontrées :
Pour la partie Frontend je n'ai pas rencontré de difficulté particulière.

Component qui me semble optimale :

```
function LocationMarker(props) {
    const [bbox, setBbox] = useState([]);

    const map = useMap();


    useEffect(() => {
        map.locate().on("locationfound", function (e) {
            props.setCurrentPosition(e.latlng);
            map.flyTo(e.latlng, map.getZoom());
            // const radius = e.accuracy;
            // const circle = L.circle(e.latlng, radius);
            // circle.addTo(map);
            setBbox(e.bounds.toBBoxString().split(","));
        });
    }, [map]);

    return props.currentPosition === null ? null : (
        <Marker position={props.currentPosition} icon={defaultIcon}>
            <Popup>
                Vous êtes ici. <br />
                Map bbox: <br />
                <b>Southwest lng</b>: {bbox[0]} <br />
                <b>Southwest lat</b>: {bbox[1]} <br />
                <b>Northeast lng</b>: {bbox[2]} <br />
                <b>Northeast lat</b>: {bbox[3]}
            </Popup>
        </Marker>
    );
}
```

Le component LocationMarker me paraît optimale car il n'est pas difficile à comprendre, il court en termes de lignes de code et pourtant il est très puissant et permet de détecter la position actuelle de l'utilisateur de notre web app.
Dans ce component j'utilise une useEffect ce qui me semble être la bonne solution ici. On essaie de détecter la position actuelle de l'utilisateur une fois que React a mis à jour le DOM et que la map est rendu(rendered).

Fonction qui aurait besoin d'être améliorée :

```
function sortTable() {
        var table, rows, switching, i, x, y, shouldSwitch;
        table = document.querySelector(".styled-table");
        if (table) {
            switching = true;
            while (switching) {
                switching = false;
                rows = table.rows;
                if (props.search) {
                    for (i = 1; i < (rows.length - 1); i++) {
                        shouldSwitch = false;
                        x = rows[i].getElementsByTagName("TD")[2];
                        y = rows[i + 1].getElementsByTagName("TD")[2];

                        x = parseFloat(x.innerHTML.split(' km')[0].split('<b>')[1]);
                        y = parseFloat(y.innerHTML.split(' km')[0].split('<b>')[1]);
                        if (x > y) {
                            shouldSwitch = true;
                            break;
                        }
                    }

                } else {
                    for (i = 1; i < (rows.length - 1); i++) {
                        shouldSwitch = false;
                        x = rows[i].getElementsByTagName("TD")[1];
                        y = rows[i + 1].getElementsByTagName("TD")[1];

                        x = parseFloat(x.innerHTML.split(';">')[1].split("</span>")[0])
                        y = parseFloat(y.innerHTML.split(';">')[1].split("</span>")[0])
                        if (x > y) {
                            shouldSwitch = true;
                            break;
                        }
                    }
                }
                if (shouldSwitch) {
                    rows[i].parentNode.insertBefore(rows[i + 1], rows[i]);
                    switching = true;
                }
            }
        }
    }
```

Cette fonction permet de trier le tableau. Cette dernière peut être amélioré en utilisant un algorithme de tri plus performant comme Bubble sort.