INGENIEUR EN SCIENCES INFORMATIQUES

RAPPORT DU PROJET WEB

Etudiant : Jonas Vihoalé ANIGLO
(SI5- Parcours AL)

Responsables : Peter Sanders, Hugo Mallet

Table des matières

1. Identifiant GitHub

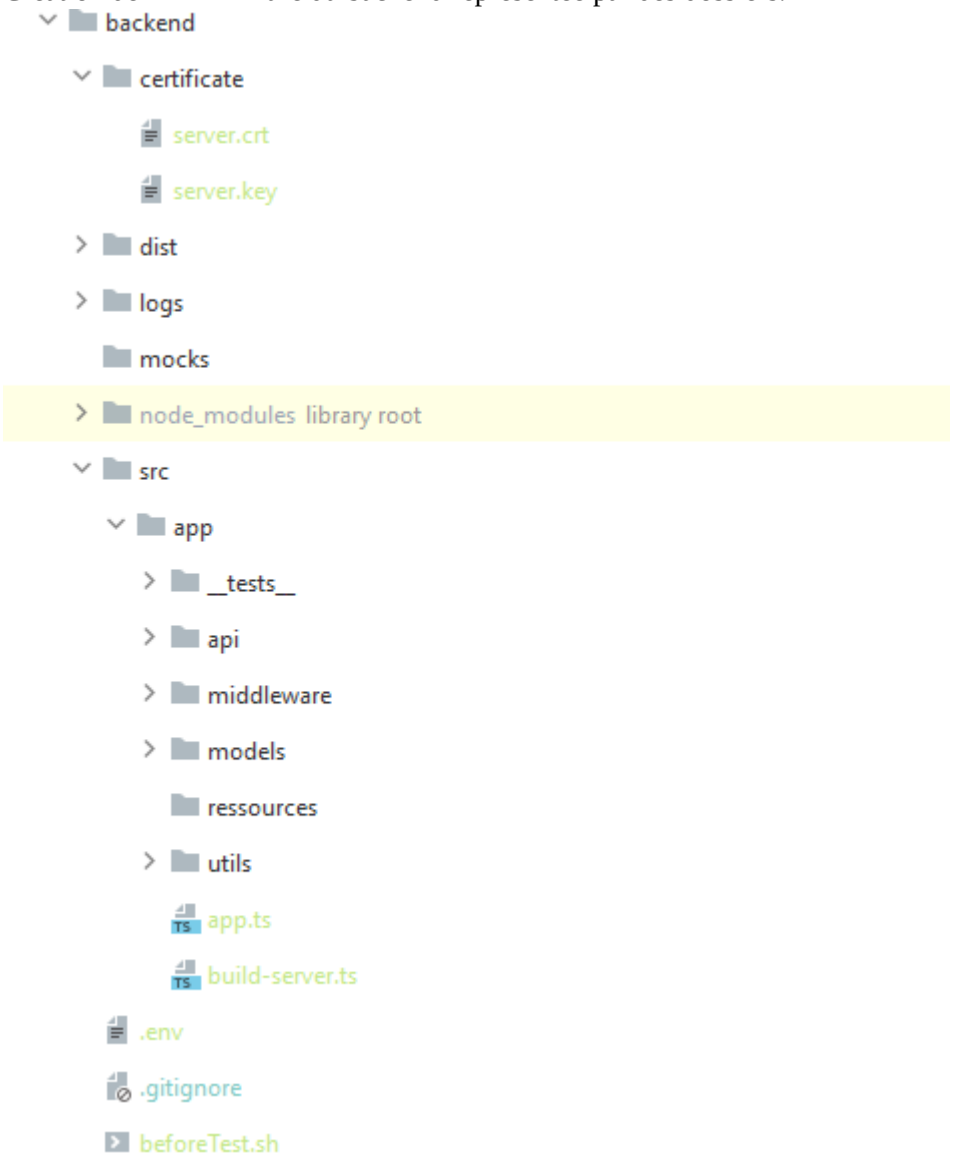
Vous pouvez accéder au projet via ce lien ainsi qu'à mon espace GitHub (JIV-DLS) via ce lien.

2. Tâches effectuées

Au cour de ce projet j'ai dédié une grande partie de mon temps au back-end, j'ai ensuite implémenté la plupart de ces fonctionnalités sur le front-end.

Fonctionnalités Backend implémentées :

- Initialisation du projet :
  Création de l'architecture du backend représentée par des dossiers.



  Certificate qui contient les certificats pour lancer le serveur en mode crypté.
  __tests__ contient tous les tests qui doivent être lancés sur le serveur.
  Api contient les différentes routes de l'API.
  Middleware contient les différents middleware notamment lorsque l'application cliente essaie d'accéder ou de modifier des ressources protégées.
  Ressources contient des fichiers ressources dont aura besoin l'application cliente. Ces fichiers peuvent être par exemple des images prise dans une station à essence.

Utils contient tout ce qui est logger et gestion d'erreur, il s'agit des classes et constantes qui peuvent être utiliser dans les autres composants.
App.ts qui importe le serveur depuis build-server puis le lance,
.env contient les informations concernant la base de donnée.
Configuration Typescript.

- Mise en place de la connexion avec Mongodb :
  Installation de mongoose, et utilisation du fichier .env pour la configuration de la base de donnée, mise en place du modèle de donnée(stations, utilisateurs …)
- Mise en place des tests pour assurer le bon fonctionnement du serveur avant chaque lancement :
  Test sur la route status de l'api
- Mise à jour du package.json pour rajouter les scripts nécessaires pour la compilation du typescript vers le js, le lancement en mode debug, l'exécution des tests avant le lancement du serveur.
- Authentification :
  Implémentation des routes nécessaires à l'authentification, il s'agit tout d'abord du login et du subscribe. L'appel à l'une de ces deux routes retourne un token contenant le userId de l'utilisateur puis une validité en temps, ce token est cypté en utilisant ***jwt*** . Dans un second temps une autre route a été créée pour la récupération des informations concernant l'utilisateur à savoir le nom prénom.
  Implémentation de deux middlewares permettant d'authentifier une requête, le premier middleware était primordiale pour déterminer si l'utilisateur est authentifié et le second utilisé pour savoir si l'utilisateur essaie d'accéder aux informations qui ne concernent que lui. Les deux middlewares ont été utile pour la vérification de la requête lorsque le client veut récupérer les informations personnelles de l'utilisateur connecté.
- Mise en place d'un scheduler qui permet de télécharger la mise à jour des données de stations depuis l'API
- Mise à jour des prix :
  Implémentation d'une route avec des middleware d'authentification permettant de modifier le prix dans une station.

Fonctionnalités Front implémentées :

- Authentification :
  Appel au backend pour la connexion et l'inscription de l'utilisateur , sauvegarde de tokens, récupération des informations personnels de l'utilisateur, redirection sur la page d'accueil
- Centralisation des appels vers le backend :
  Mise en place d'un module appelé API qui contient des méthodes pour les différents appels vers le backend. Ce module initialise un objet avec Axios. L'avantage était de tout centralisé. Les appels nécessitant l'authentification se servent d'une méthode de configuration qui récupère le token stocké dans le localStorage.
- Authentification des utilisateurs:
  Appels effectués vers le backend pour le login et le subscribe. Une fois l'authentification faite une redirection est réalisée sur la page d'accueil qui affiche le composant principale. De là un test est effectué pour savoir si le token existe, s'il existe une requête est alors faite pour récupérer les information de l'utilisateur. Ce test et cet appel sont fait depuis un UseEffect pour ne faire l'appel qu'une seule fois quand le composant est chargé. Une fois les informations des utilisateurs récupérées, un state(user) est utilisé pour sauvegarder les données récupérées et les partager avec les composants fils. Les composants fils ne sont pas très profond(le composant parent n'a au plus que 2 composants fils) sinon au lieu des states j'aurai pu utiliser des Contexts.

- Modification du prix par un utilisateur :
  Récupération des données de mise à jour par rapport à un prix et envoi d'une requête Put au backend pour mettre à jour le prix, une fois fait un alert est fait indiquant à l'utilisateur si la requête est passée ou pas
- Récupération de la position actuelle de l'utilisateur :
  Utilisation d'un useEffect pour ne demander qu'une seul fois au démarrage la position de l'utilisateur, la callback envoyer une fois la récupération effectué permet de sauvegarder la position dans un state partager avec le composant fils affichant la carte. Cette position est ensuite affiché avec un marqueur et une icône.

3. Organisation du groupe (GitHub)

Afin de nous départager les tâches équitablement, nous avions créées des issues sur le GitHub du projet. Nous pouvions alors y retrouver toutes les tâches qui restait à faire, ainsi que savoir ce que d'y ajouter des nouvelles tâches lorsqu'il était nécessaire.

4. Difficultés rencontrées

Dans le backend j'ai eu beaucoup du mal à trouver les bibliothèques nécessaire en typescripts, il s'agit notamment de mongoose, après cela la mise en place des tests ne marchait pas car le serveur à son démarrage lançait le scheduler qui devait charger les mise à jour à 6h du matin, ce qui empêchait l'arrêt des test.

Du côté frontend, l'appel à des méthodes qui prenait un temps considérable tel que les appels backend des modification de la carte faite directement dans les classe fesait que ces derniers était fait à chaque rerender, ce qui rendait très lent l'application et causait une exception de nombre maximum de re-render parce que des appels recursifs se produisaient des fois. La prise en main des state et le passage en paramètres au composants enfants qui ne marchaient pas.

5. Commentaires sur le code

Code élégant / optimal : API
La classe API fesant des appels au backend, il contient toutes les informations nécessaires pour toutes les requêtes. Ces informations sont les configurations des requêtes. Utilisant Axios on peut rajouter des configurations tels que l'Autorisation, la base API(le lien vers l'API backend), le timeout

Code Méritant une optimisation :

- La sauvegarde des informations clientes

Il peut être optimisé en sauvegardant l'essentielles des informations de l'utilisateurs dans le localStorage.

- Le passage des infos des infos utilisateurs aux composants enfants.
  Il pourra être optimiser en utilisant un context qui fournira ces infos.
- Récupération de la position actuelle de l'utilisateur

La version actuelle consiste a récupérer la position de l’utilisateur qu’au démarrage de l’application. Une amélioration pourrait être faite en demander par intervalles de temps la position de l’utilisateur