La structure de fichiers d'un projet React

Objectifs

Comprendre à quoi servent les fichiers et les dossiers créés.

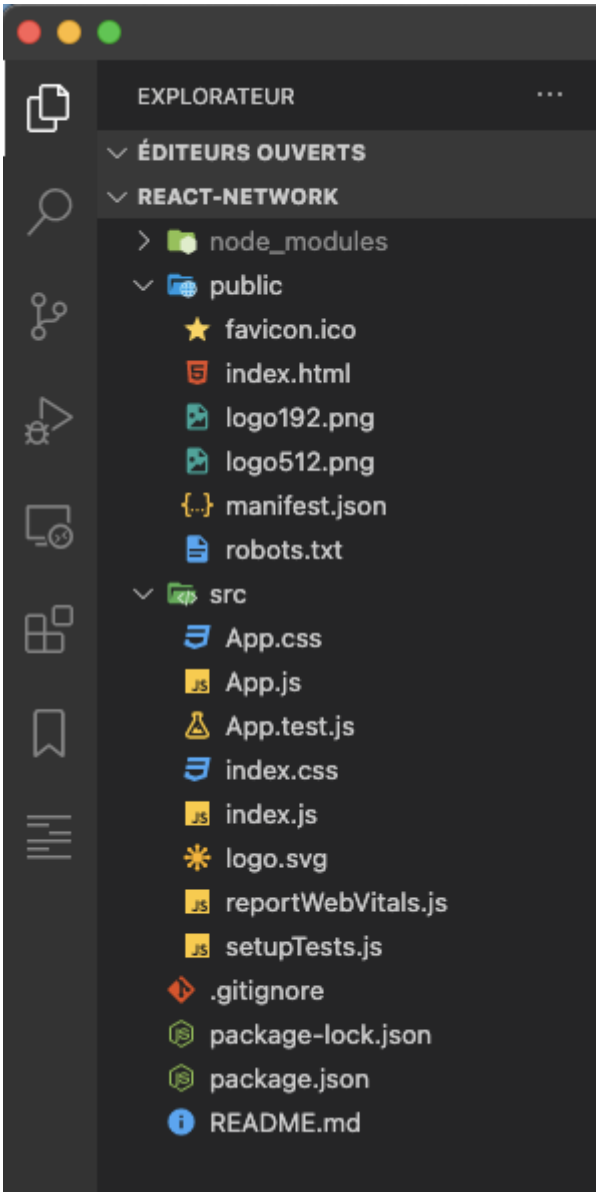
Comprendre

Lorsque vous allez ouvrir votre projet, vous aurez peut-être des fichiers ou des dossiers que je n'ai pas, ou inversement.

En effet, l'utilitaire create-react-app évolue en permanence, des nouvelles fonctionnalités sont ajoutées en permanence.

Si vous avez des fichiers en plus ou en moins, laissez-les tel que vous les avez, cela ne devrait poser aucun problème.

Si on déplie les dossiers `/public` et `/src`, voici à quoi devrait ressembler votre projet dans VS Code :



Analysons chacun des fichiers (pas forcément dans l'ordre d'apparition ni alphabétique) :

- `README.md` : Ce fichier contient la description du projet au format markdown. C'est le contenu de ce fichier qui sera affiché sur votre page d'accueil github si vous utilisez git et github (hors du cadre de cette formation)

- `package.json` : Il contient la version de chaque package npm installé. On y trouve aussi des alias de commandes comme `npm start`, `npm build`, etc. qui permettront de facilement démarrer ou builder notre application.

- `package-lock.json` : Il ressemble à package.json mais il garde une trace de la version exacte installée. Par exemple on pourra avoir une version `1.0.*` dans packge.json pour dire qu'on installa la dernière version 1.0 disponible, mais si en l'occurence la dernière version 1.0 disponible est la `1.0.6`, alors on va tracer dans package-lock.json qu'on a installé la `1.0.6`. Le but est que tous les développeurs d'une équipe utilisent les mêmes versions.

- `node_modules` : lorsqu'on fait `npm install`, npm télécharge tous les packages listés dans package.json depuis le web, et les stocke dans node_modules.

  Ce dossier peut donc devenir énorme (parfois des Go sur de gros projets). Cela permet de comprendre l'utilité du package.json lorsqu'on utilise des logiciels de gestion de versions comme git : au lieu de commiter les librairies elles-mêmes (node modules, potentiellement plusieurs Go de données), on va simplement commiter les fichiers package.json et package-lock.json qui donnent la liste des librairies à télécharger.

  Dans notre repository git, on a donc un fichier package.json de quelques ko et pas des Go de node_modules. Le projet sur git est donc de taille réduite, rapide à téléchargé, et une fois qu'il est téléchargé sur le poste d'un développeur, ce dernier va lancer le téléchargement des librairies listées dans le package.json grâce à la commande `npm install`.

- `.gitignore` : dans le contexte d'un projet git, ce fichier va lister les fichiers et dossiers de notre projet que l'on ne souhaite jamais commiter (comme par exemple le dossier `node_modules`)

- `public` : Ce dossier contient les ressources statiques de notre application (le squelette HTML). Tout ce qui est dans ce dossier n'a rien à voir avec React, on pourrait avoir les mêmes fichiers dans un simple projet HTML :

  - `index.html` : Le fichier html qui accueillera notre application React. Il contient une balise `<div id='root'></div>` : c'est à l'intérieur de cette balise que React va injecter notre application complète.

    - `favicon.ico` : Le favicon de notre application
    - `robots.txt` : Un fichier pour dire aux moteurs de recherche ce qui peut être indexé ou pas
    - `manifest.json` : Un fichier qui permet de facilement convertir notre application en PWA (progressive web app)
    - `logo192.png` et `logo512.png` : Des images listées dans le manifest.json pour une éventuelle utilisation PWA

- `src` : Le dossier le plus important : c'est ici que se trouve le code de notre application React. Plusieurs fichiers ont été créés pour générer une application de démo. Expliquer le contenu de ces fichiers reviendrait à prendre trop d'avance sur la formation, on va donc supprimer l'appli de démo pour ne garder que le squelette le plus simple possible et partir de zéro.