esprit Se former autrement HONORIS UNITED UNIVERSITIES	Année Universitaire : 2022-2023 

Atelier n°0 :

Installation et configuration d'une application React

Objectifs :

- Configuration d'un environnement de travail local pour React.
- Créer un projet **React** tout en utilisant l'outil **CRA** (**Create React App**).
- Manipuler les fonctionnalités de Ecmascript 6 (**ES6**).

Contexte de l'atelier :

Au cours de cet atelier, nous allons configurer l'environnement de travail et créer un nouveau projet React à l'aide de l'outil CRA. Nous allons ensuite essayer de manipuler les fonctionnalités de ES6.

Partie 1 : Configuration de l'environnement de travail

Installer Node JS, un environnement d'exécution JavaScript backend qui nous permettra d'utiliser un gestionnaire de packages nommé NPM et donc, le package runner NPX.

Lien pou Node JS : https://nodejs.org

1. Création du projet :

Une fois l'environnement Node Js installé, vous devriez pouvoir créer un projet à l'aide de Npx.

- Assurez-vous d'être connecté à un réseau internet avant de lancer les commandes NPM/NPX.

a) Positionnez-vous au niveau du dossier dans lequel vous souhaitez créer votre projet et exécutez l'une des commandes suivantes :

Avec NPM:

```
npm create-react-app workshop-react
```

Tel un énorme annuaire, NPM permet de télécharger, installer, mettre à jour et désinstaller les packages relatifs à l'environnement JavaScript.

Avec NPX:

```
npx create-react-app workshop-react --use-npm
```

Avec YARN:

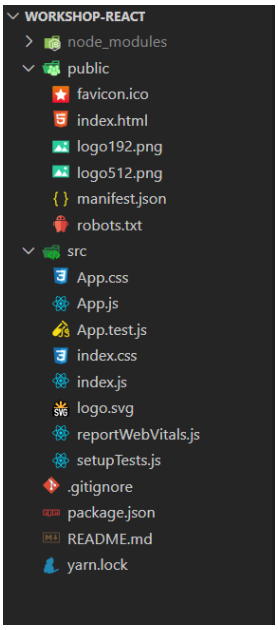
```
yarn create react-app workshop-react
```

- NB: Le nom du projet ne doit contenir ni des caractères spéciaux, ni des majuscules.

Après avoir créé votre projet, vous pourrez l'exécuter avec la commande suivante : **yarn start ou npm start.**

Ouvrir le nouveau dossier créé avec **Visual Studio Code.**

Votre projet devrait avoir la structure suivante :



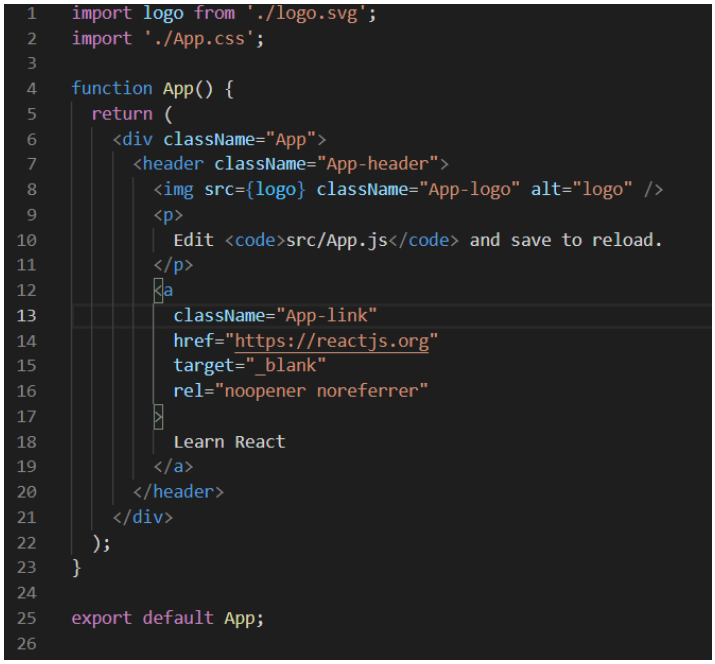
- Prenez note que si vous utilisez npm/npx « yarn.lock » sera remplacé par « package.lock ».

node_modules : contient tous les packages nécessaires au fonctionnement de notre projet.

Public : contient les informations générales de notre application.

Src : contient la logique métier de notre application:

- **SetupTests**: Initialise l'environnement de test
- **App.test:** Contient les tests unitaires par défaut de l'App component
- **ReportWebVitals.js** : aide à mesurer les performances de votre application à l'aide des outils Chrome web_vitals.
- **Index.js** : le point d'entrée de notre application.
- **App.js** est le component initial :



```
import logo from './logo.svg';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.js</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;

```

- **Package.json** : contient les informations et les descriptions des dépendances relatives au projet..
- **Yarn.lock/package.lock** : décrit la dernière bonne configuration connue pour une application donnée.

Partie 2 : EcmaScript (ES6)

ES6 est la version récente de JavaScript qui apporte de nombreuses améliorations, notamment des fonctionnalités supplémentaires dans la bibliothèque standard et des syntaxes renouvelées. Au cours de cet atelier, nous allons explorer certaines des fonctionnalités les plus récentes de cette version.

Pour commencer, créez un nouveau fichier js dans votre projet sous un dossier appelé **Ecmascript** sous **src** et résolvez les problèmes suivants :

1. Créez une fonction appelée "findLongestWord" qui prend en paramètre un tableau de mots. La fonction doit retourner le mot le plus long du tableau, ainsi que sa longueur.

Utilisez la syntaxe ECMAScript 6, y compris :

- **L'affectation par décomposition pour attribuer le tableau de mots à une variable.**
- **La méthode .map() pour créer un nouveau tableau d'objets, où chaque objet a une propriété "mot" et une propriété "longueur"**
- **La méthode .reduce() pour trouver l'objet avec la plus grande propriété "longueur".**
- **Le mot-clé let pour déclarer des variables.**
- **Une fonction flèche pour définir la fonction.**

2. Comptez les occurrences d'éléments distincts dans un tableau donné. L'entrée donnée est un tableau, dont les éléments sont **des tableaux de chaînes de caractères**.

   Le résultat est un objet dont **les noms des propriétés** sont les valeurs des tableaux et **leur valeur** est le nombre de leurs occurrences.

Utilisez les méthodes : .reduce() et .flat()

Input :

```
const input = [
    ["a", "b", "c"],
    ["c", "d", "f"],
    ["d", "f", "g"],
  ];
```

Output :

```
{
  a: 1,
  b: 1,
  c: 2,
  d: 2,
  f: 2,
  g: 1,
}
```

3. Calculez le total des notes des élèves ayant obtenu une note supérieure à 50 après avoir ajouté un bonus de 15 points aux élèves ayant obtenu une note inférieure à 50.

Utilisez les méthodes : .map(), .filter() et .reduce()

Input :

```
let students = [{ name: 'John', id: 123, marks: 98 },
{ name: 'Baba', id: 101, marks: 23 },
{ name: 'John', id: 200, marks: 45 },
{ name: 'Wick', id: 115, marks: 75 },];
```

Output :

```
233
```

4. Créez un tableau d'objets appelé **"Tab"** contenant des données de votre choix.
    - Ajoutez de nouvelles entrées de votre choix en utilisant la méthode "push" ou "unshift".
    - Ajoutez une propriété ID à chaque objet dans le tableau.

Remarque : Utilisez une variable globale pour stocker la dernière valeur ID utilisée, et l'incrémenter à chaque fois.

- Créez un nouveau fichier js appelé **fonction.js** sous le dossier **Ecmascript** qui contient une fonction **Search** de recherche par **id** d'un objet bien déterminé à partir d'un tableau.
- Exportez la fonction **Search** et l'utiliser avec le tableau **Tab** déjà crée afin de chercher un élément voulu.