3. Comprendre les concepts de React JS

3.1. Présentation de React :

React (Ou ReactJS, React.js) est une bibliothèque JavaScript développée par Facebook, utilisée pour créer des com- posants d'affichage réutilisables.

Pour créer une application avec React, on va donc créer des composants d'affichage (en général des classes ou des fonctions) qui seront ensuite assemblés afin de former l'application finale. L'intérêt de React, outre le fait de créer des composants réutilisables, et de permettre aussi une mise à jour rapide de la page HTML.

En effet, lors de l'utilisation de bibliothèques telles que jQuery (qui manipulent les éléments HTML de la page, c'est-à-dire le DOM, directement), les performances sont bien moindres car la manipulation du DOM directement en JavaScript est lente lors de l'exécution. React ne manipule pas le DOM directement mais une copie interne de celui-ci (les objets ou éléments React, appelés le DOM virtuel), et produit les modifications sur l'affichage uniquement lorsque cela s'avère nécessaire.

Enfin, React n'est pas seulement une bibliothèque permettant d'effectuer des affichages sur le Web, elle peut également servir à produire des applications natives iPhone et Android, en utilisant une variante nommée React Native.

3.2. L'écosystème React :

NPM

npm fait deux choses : gérer vos dépendances et lancer des scripts

La gestion de dépendance vous assure que les librairies que vous utilisez sont à jour. Si vous avez un background plus orienté back-end, c'est l'équivalent de composer pour PHP, maven pour Java, encore pip pour Python, ou encore Bundler pour Ruby. Côté front-end, npm c'est beaucoup plus suivi par la communauté React, et il gérera aussi toutes vos dépendances en node.

Le lanceur de script exécute les différentes tâches dont vous avez besoin dans votre projet : la compilation, le démarrage de l'application, etc. C'est aussi un excellent moyen d'unifier les projets en proposant des standards valables sur tous les projets.

Quand vous récupérez un projet dans le Grand Internet, généralement vous aurez les commandes suivantes :

- **npm install** : installe le projet sur votre machine.
- **npm start** : démarre le projet.
- **npm test** : lance les tests du projet.
- **npm run dev** : s'occupe de lancer un environnement de développement agréable.

ECMAScript6

ECMAScript est un standard de langage de programmation, il définit : La syntaxe, Les types de variable et encore pas mal d'autres choses…

ES6 : Il a été publié en 2015, d'ailleurs vous pouvez aussi le voir aussi sous le nom ES2015. Beaucoup de choses ont evolué en ES6… **Fonctions Fléchées**, **Classes**, **Modules**…

Babel

Afin de faire de l'ES6 dès aujourd'hui, il faut le compiler, ou le transpiler. Pour cela il y a notamment **Babel** qui va transformer pour vous le code en ES5 (version qui est supportée par tous les navigateurs modernes). Il peut aller plus loin que de l'ES6 et a tendance à proposer des solutions via un système de plugin pour les propositions qui sont parfois encore en draft.

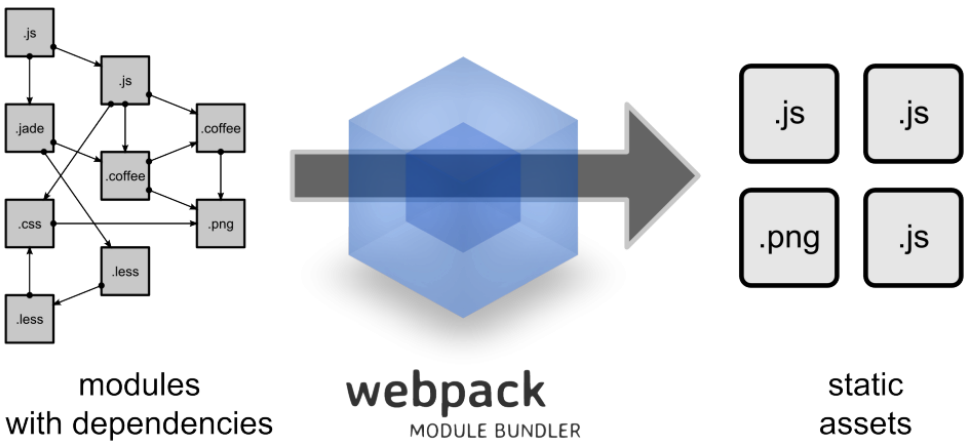
Babel fonctionne avec des presets, qui sont un ensemble de plug-ins permettant de compiler toutes les tâches liées à la technologie qu'on voudra compiler. En ce qui nous concerne, on aura besoin des presets suivants : babel-preset-es2015 et babel-preset-react.

Bundlers

Le bundler condense tout le code de votre application pour n'avoir qu'un seul fichier javascript à importer dans votre page HTML. L'intérêt majeur c'est que vous n'êtes plus contraints d'avoir un fichier obèse quand vous codez du JavaScript, où il est impossible de retrouver votre fonction. Vous pouvez faire des modules (fichiers isolés), y faire référence dans un autre fichier et tout de même être capable de les utiliser dans le navigateur sans ajouter une balise <script> à chaque fois que vous ajoutez un fichier.

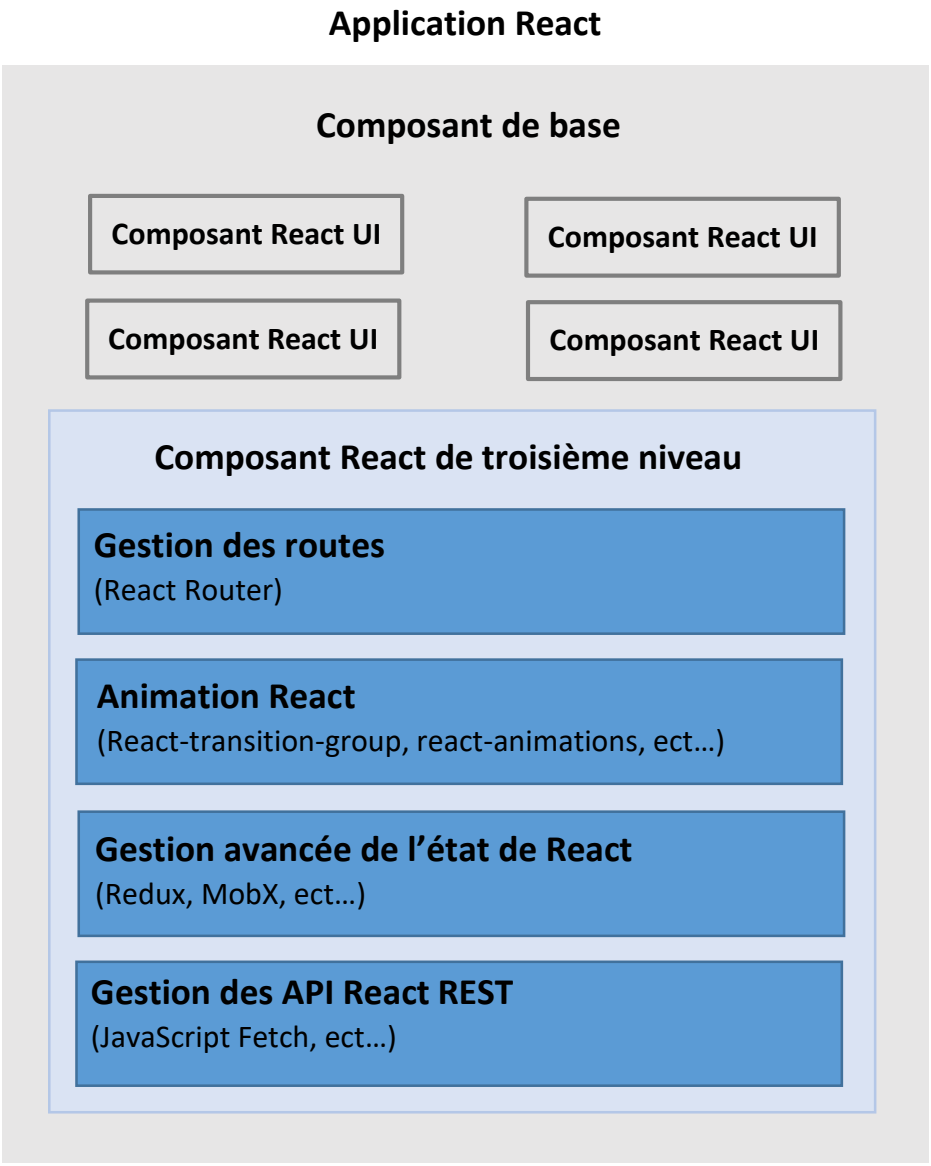
Les deux outils majeurs sur le marché sont Browserify et Webpack.

Webpack est un outil qui permet de prendre en compte la modularité du code JavaScript. Il peut le faire selon les différents standards (commonJs, AMD, etc.). En revanche, à lui tout seul, il est incapable de transformer le code ES2015 ou JSX.



3.3. Architecture d'une application React :

Afin d'écrire un bon code, la bibliothèque React propose différents concepts comme : **le composant d'ordre supérieur, le contexte, les accessoires de rendu, les références**, etc. **React Hooks** est utilisé pour faire gérer de grands projets. Examinons l'architecture d'une application React :

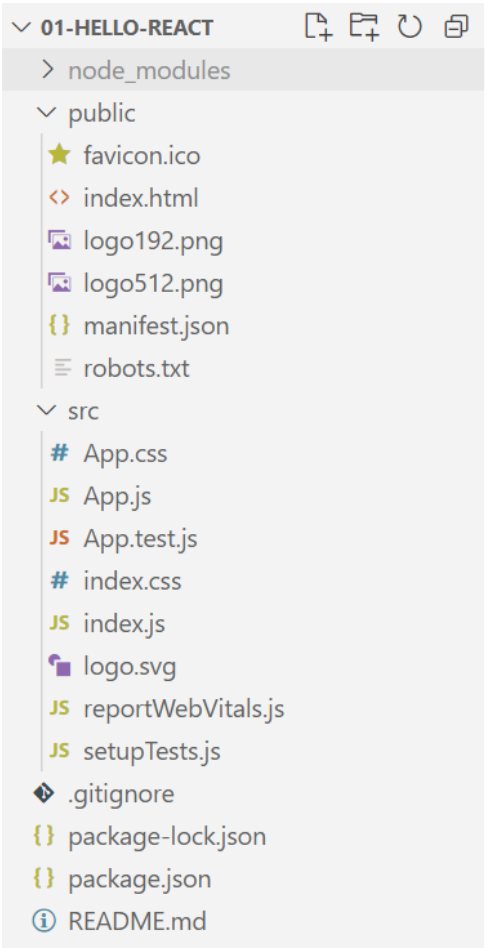


1. L'application React commence à démarrer avec un seul composant racine ;
2. Le composant racine est créé grâce à un ou plusieurs composants ;
3. Chaque composant est formé comme un composant de composants plus petits au lieu de dépendre d'un autre composant ;
4. La grande partie des composants sont des composants d'interface utilisateur ;
5. L'application React a la possibilité d'ajouter un composant tiers à des fins spécifiques comme le routage, l'animation, la gestion de l'état, etc.

Créer votre toute première application React.js *(Voir le document des travaux pratiques)*

Structure de projet:

```
01-HELLO-REACT
├── README.md
├── node_modules
├── package-lock.json
├── package.json
├── .gitignore
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── reportWebVitals.js
    └── setupTests.js
```



Le dossier **node_modules** contient toutes les librairies javascript dont vous aurez besoin

Le dossier **public** vous y trouverez le fichier index.html qui est notre point départ avec la div qui a pour class root qui permet de signifier a React ou est ce qu'il doit venir s'ancrer.

Le fichier **.gitignore** a pour rôle de signifier a git les fichiers qui ne doivent pas être pusher sur le repo distant notamment le dossier node module.

Le fichier **package.json** est certainement le fichier qui contient plusieurs informations sur votre application React notamment les scripts, les dépendances.

Le fichier **Readme** permet de documenter votre app en Markdown.

Le dossier **src** est l'endroit où toute la magie de votre application va opérer, on y retrouve le fichier index.js qui est le fichier qui charge notre premier composant le composant App qui est ensuite greffer a la div avec la class root qui se trouve dans le fichier index.hmtl au niveau du dossier public.

3.4. Explication du code :

Quand vous créez un projet React avec **create-react-app**, il y a deux fichiers principal, le fichier **public/index.html** et le fichier **src/index.js**

Commençons par le fichier **public/index.html** :

Ce fichier ne contient pas grand-chose, c'est là qu'est défini la structure de base d'une page HTML, plus important il y est défini l'id root

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <title>React App</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this
app.</noscript>
    <div id="root"></div>
  </body>
</html>
```

C'est en effet dans cette div que notre application sera afficher.

Passons ensuite au fichier **src/index.js** :

Ce fichier va permettre de récupérer nos composants et les afficher dans la div#root de public/index.html

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5  import reportWebVitals from './reportWebVitals';
   const root =
   ReactDOM.createRoot(document.getElementById('root'));
   root.render(
6    <React.StrictMode>
       <App />
     </React.StrictMode>
   );
7  reportWebVitals();
```

La ligne 1 : importe la bibliothèque React elle-même. Comme React transforme l'instruction JSX que nous écrivons en React.createElement(), tous les composants React doivent importer le module React. Si vous ignorez cette étape, votre application produira une erreur.

La ligne 2 : importe Le module react-dom qui fournit des méthodes spécifiques au DOM que vous pouvez utiliser au niveau racine de votre application et comme échappatoire pour travailler hors du modèle React si vous en avez besoin.

La ligne 3 : importe la feuille de style **src/index.css**

La ligne 4 : importe le fichier **src/App.js** qui définit une fonction nommée **App.**

La ligne 6 : affiche l'élément React **<App />** au sein du nœud DOM **root** et renvoie une référence sur le composant.

Ligne 5 et 7 : Web Vitals sont un ensemble de mesures utiles qui visent à capturer l'expérience utilisateur d'une page Web.

Avec la fonction reportWebVitals, vous pouvez envoyer n'importe résultats à un service d'analyse d'audience comme Google Analytics.

Passons maintenant au composant App (**src/App.js**) :

```
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;
```

La fonction **App** renvoie une expression JSX. Cette expression définit ce que votre navigateur restitue finalement au DOM.

Tout en bas du fichier **App.js**, l'instruction ***export default App*** rend notre composant App disponible pour les autres modules.