COURS REACT JS

MOURAD ARAB

Table des matières

INTRODUCTION

Depuis quelques années, l'utilisation des frameworks pour le développement des applications web est devenu un incontournable. Un des frameworks les plus populaires de nos jours est React.

React est une librairie Javascript open source développée par Facebook, dont la première release a vu le jour au début de l'année 2013. Aujourd'hui, elle a été mise en place sur des plateformes célèbres telles que Netflix, Yahoo, AirBnb et autres grandes enseignes.

Cette librairie a plusieurs avantages, parmi elles :

Elle est open source : vous pouvez contribuer à l'amélioration de la librairie en signalant des problèmes ou en créant des demandes de tirage. *(Il vous suffit de suivre cette documentation !)*

- Elle est déclarative : vous écrivez le code que vous souhaitez, React prend ce code déclaré et effectue toutes les étapes JavaScript/DOM pour obtenir le résultat souhaité.

- Elle est basée sur les composants : les applications sont créées avec des modules de code indépendants préfabriqués et réutilisables qui gèrent leur propre état et peuvent être collés ensemble à l'aide du framework React, ce qui permet de transmettre des données à votre application tout en conservant l'état hors du DOM.

- JSX est une extension de syntaxe pour JavaScript, la connaissance du javascript facilite la prise en main de React.

PARTIE 1 - DECOUVERTE

Premières manipulations

Dans notre fichier html nous utilisons le code JavaScript pour mettre à jour le DOM html et ajoutons une balise h1 avec le texte « Bonjour» dans la balise div d'id root.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="root"></div>
    <script>
        const root = document.getElementById('root');
        console.log(root);
        let h1 = document.createElement('h1');
        h1.innerText = "Bonjour";
        root.appendChild(h1);
    </script>
</body>
</html>
```

Tester React

Pour tester React rapidement, il suffit d'ajouter les balises script suivantes dans un code html avant la balise fermante </body> d'un fichier html.

```
    <!-- ... autres contenus HTML ... -->

    <!-- Charge React -->
    <!-- Remarque : pour le déploiement, remplacez "development.js"
         par "production.min.js" -->
    <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>

    <!-- Charge notre composant React -->
    <script src="like_button.js"></script>
</body>
```

Nous exécutons la même tache que précédemment mais cette fois ci avec React. Nous ajoutons donc une balise h1 avec le texte « Bonjour » dans la balise div d'il root.

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8  </head>
9  <body>
10     <div id="root"></div>
11     <script>
12
13         const root = document.getElementById('root');
14         let h1 = React.createElement('h1', null, "Bonjour2");
15         ReactDOM.render(h1,root);
16
17     </script>
18     <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
19     <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
20 </body>
21 </html>
```

Analyse :

Dans cet exemple de code nous récupérons l'élément div d'id root de la ligne 10 dans une constante nommée « root »(ligne 13).
En ligne 14 nous créons un élément **h1** auquel nous définissons un texte « Bonjour2 », que nous affectons à la variable h1 puis,
En ligne 15 à l'aide de la méthode 'render' nous chargeons cette élément h1 dans la constante root définie en ligne 13.
Le navigateur affiche une balise div qui contient la balise h1 définie.

3.Construire une page
Construire une page devient vite fastidieux avec la méthode précédente. Voici un exemple montrant l'aspect laborieux.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
</head>
<body>
    <div id="root"></div>
    <script>

        const root = document.getElementById('root');
        const para = React.createElement('p', {children: 'Voici du text'});

        let h1 = React.createElement('h1', {children: para});
        console.log(h1);
        ReactDOM.render(h1, root);

        //devient compliqué
        <baliseprincipale>
            <section>
                <h1>Nouveau Titre</h1>
            </section>
        </baliseprincipale>

    </script>
</body>
</html>
```

Dans cette exemple nous définissons toujours à la ligne 15 une constante nommée « root » puis une constante nommée « para » qui contient l'élément p. Grace à createElement() , en 1er paramètre nous indiquons la balise, et en 2ieme argument un « props » qui définit en réalité un ensemble d'argument représentés par le concept clés - > valeur.
La clé « children » donne accès au texte « voici du text ».
En faisant un console.log(para.props.children), nous obtenons dans la console « Voici du text ».
Ligne 18 nous créons un composant h1 qui contiendra le composant 'p' de para.
Puis ligne 20 nous chargeons notre composant h1 dans la balise div d'id « root ».
Il devient vite pénible de construire une partie du DOM tel que la ligne 23 à 27.

JSX, une utilisation plus simple

Nous allons maintenant utiliser l'écriture JSX, qui va nous faciliter l'assemblage de nos éléments.

```html
<script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
```

Il nous suffira pour cela d'ajouter le script à notre fichier html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
  <script src="https://unpkg.com/react@16/umd/react.production.min.js" crossorigin></script>

  <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</head>
<body>
    <div id="root"></div>

    <script type="text/babel">

        const root = document.getElementById('root');

        const monModel = (
        <main>
            <section>
                <h1>Nouveau Titre</h1>
            </section>
        </main>
        );

        ReactDOM.render(monModel, root);
    </script>
</body>
</html>
```

Le code suivant nous montre à quel point il est facile dorénavant de construire notre DOM.
Après la récupération de notre div d'id root ligne 20,
Nous construisons notre pattern ligne 22 à 28, puis nous l'affichons grâce à la ligne 30.
Nous avons construit un pattern que nous avons affecté à une constante monModel.
L'affichage se fera via la méthode render.

Les Composants

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Document</title>
8
9      <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
10     <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
11     <script src="https://unpkg.com/react@16/umd/react.production.min.js" crossorigin></script>
12
13     <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
14 </head>
15 <body>
16     <div id="root"></div>
17
18     <script type="text/babel">
19
20         const root = document.getElementById('root');
21         const titre = "Nouveau Titre";
22
23         const MonComposant = function(){
24             return(
25                 <section>
26                     <h1>{titre}</h1>
27                 </section>
28             )
29         }
30         const monModel = (
31         <main>
32             <MonComposant/>
33         </main>
34         );
35
36         ReactDOM.render(monModel, root);
37     </script>
38 </body>
39 </html>
```

À la ligne 21 nous avons créé une variable ‘titre’ à laquelle nous affectons du texte. Puis à partir de la ligne 23 nous créons un composant fonctionnel que nous affectons à une constante MonComposant.
Ensuite nous utilisons notre composant ‘MonComposant’ dans le pattern de la constante monModel.
Pour finir, nous réalisons l’affichage via la méthode render.

Les props

Dans l’exemple qui suit nous intégrons le props.
Les composants sont comme des fonctions JavaScript, ils acceptent des entrées quelconques (appelées « props ») et renvoient des éléments React décrivant ce qui doit apparaître à l’écran.

Cette fois nous avons ajouté un paramètre nommé « props » à notre composant fonctionnel.

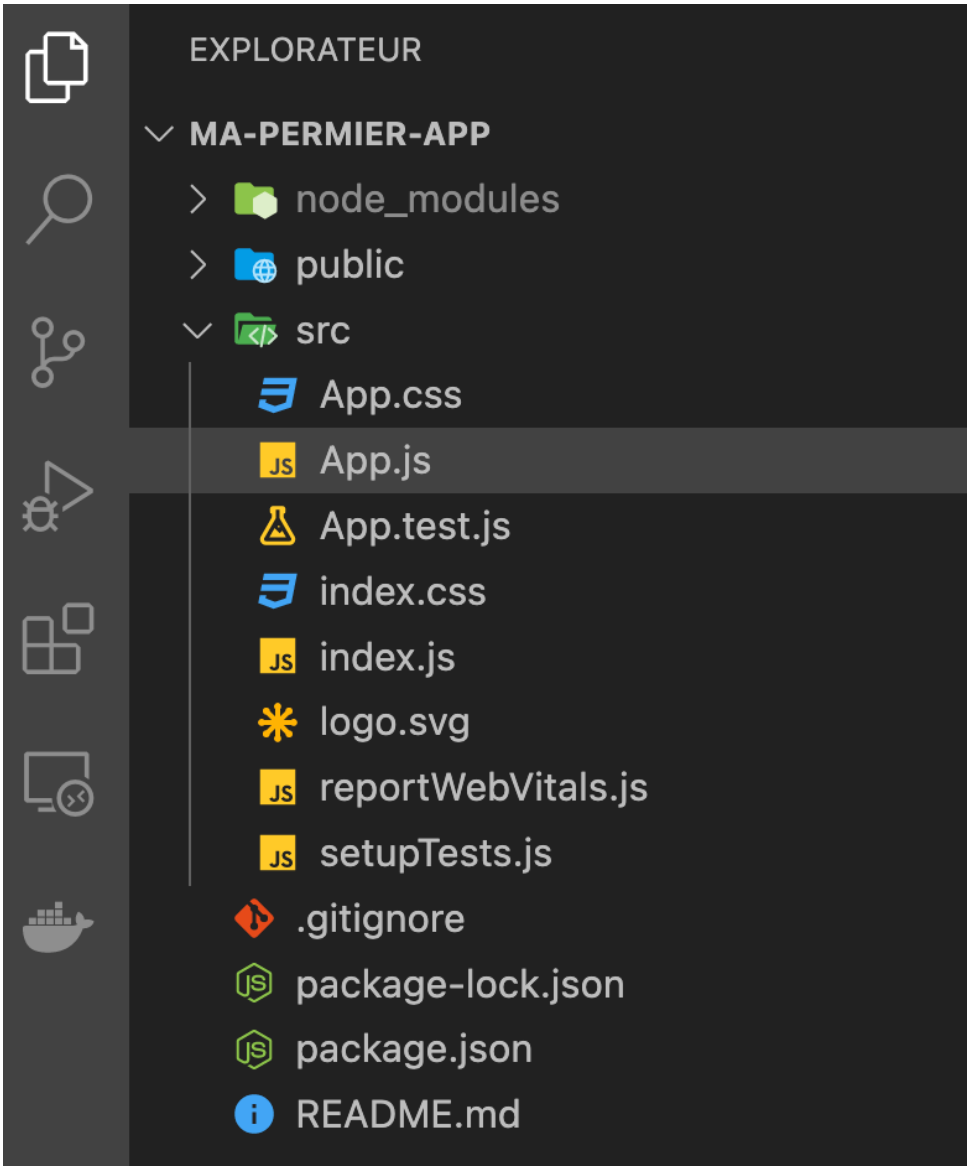
```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>Document</title>
9
10     <script src="https://unpkg.com/react@16/umd/react.development.js" crossorigin></script>
11     <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js" crossorigin></script>
12     <script src="https://unpkg.com/react@16/umd/react.production.min.js" crossorigin></script>
13
14     <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
15 </head>
16
17 <body>
18     <div id="root"></div>
19
20     <script type="text/babel">
21
22         const root = document.getElementById('root');
23
24         const MonComposant = function (props) {
25             return (
26                 <section>
27                     <h1>{props.data}</h1>
28                 </section>
29             )
30         }
31         function MonModel() {
32             const titre = "Nouveau Titre";
33             return (
34                 <main>
35                     <MonComposant data={titre} />
36                 </main>
37             )
38         };
39
40         ReactDOM.render(<MonModel />, root);
41     </script>
42 </body>
43
44 </html>
```

De cette manière nous pouvons transférer des données à notre composant comme à la ligne 35, où nous ajoutons un attribut ‘data’ auquel nous associons indirectement le texte « Nouveau Titre ».
À la ligne 27 nous exploitons cet attribut grâce à la commande suivante {props.data}.

PARTIE 2 - PREMIER PAS

Installation de mon environnement

La mise en place d’un environnement React commence par l’installation de Node.js.
Utiliser la commande dans un terminal pour initialiser un projet : ‘create-react-app’.
Vous pouvez utiliser un éditeur de texte tel que visualStudioCode.



Voici l'ensemble des fichiers générés par la commande « create-react-app » suivie du nom du projet.

-Nodes modules:

Dans l'ensemble vous n'aurez pas besoin d'ouvrir ce dossier, il contient toutes les librairies javascript dont vous aurez besoin.

-Public:

Dans le dossier public vous trouverez le fichier **index.html** qui est notre point de départ avec la div qui a pour class root qui permet de signifier a React à quel endroit il doit venir s'ancrer.

-.gitignore

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.

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

-Readme :
Le fichier Readme permet de documenter votre app en Markdown, il est fortement recommandé si vous voulez vous souvenir de tout ce qu'il y'a à faire ou si vous voulez que d'autre personnes participent au projet.

-Src:
Le dossier src est l'endroit où vous passerez le plus de temps, on y retrouve le fichier index.js qui est le fichier qui charge notre premier composant le composant App qui est ensuite greffer à la div avec la class root qui se trouve dans le fichier index.html au niveau du dossier public. Nous pouvons créer un composant de 2 manières, cela peut être un composant fonctionnel ou un composant de classe.

Composant de classe
Le composant de classe, un composant avec état/conteneur, est une classe ES6 standard qui étend la classe de composant de la bibliothèque React. Il est appelé composant avec état car il contrôle la façon dont l'état change et la mise en œuvre de la logique du composant. En dehors de cela, ils ont accès à toutes les différentes phases d'une méthode de cycle de vie React.

```
1   import React from "react";  6.9k (gzipped: 2.7k)
2
3
4   class MonComposant extends React.Component{
5   constructor(props){
6       super(props);
7       this.state = {
8           monAttribut:'',
9           textInput:'',
10      }
11      this.textChange = this.textChange.bind(this);
12  }
13
14  textChange(e){
15      this.setState({monAttribut:e.target.value})
16      }
17
18  render(){
19      return(
20      <div>
21          <input type="text" onChange={this.textChange}></input>
22          <label>{this.state.monAttribut}</label>
23      </div>
24      )
25  }
26
27  }
28
29  export default MonComposant;
```

Dans l'exemple nous créons une classe MonComposant qui hérite de Component.
Dans le constructeur ligne 5 nous avons le paramètre 'props' que nous utilisons ligne 6 lors de l'appel du constructeur de la classe mère via la commande super().
Ligne 7 nous déclarons le state initialisé avec

{

monAttribut:'',

textInput:'',

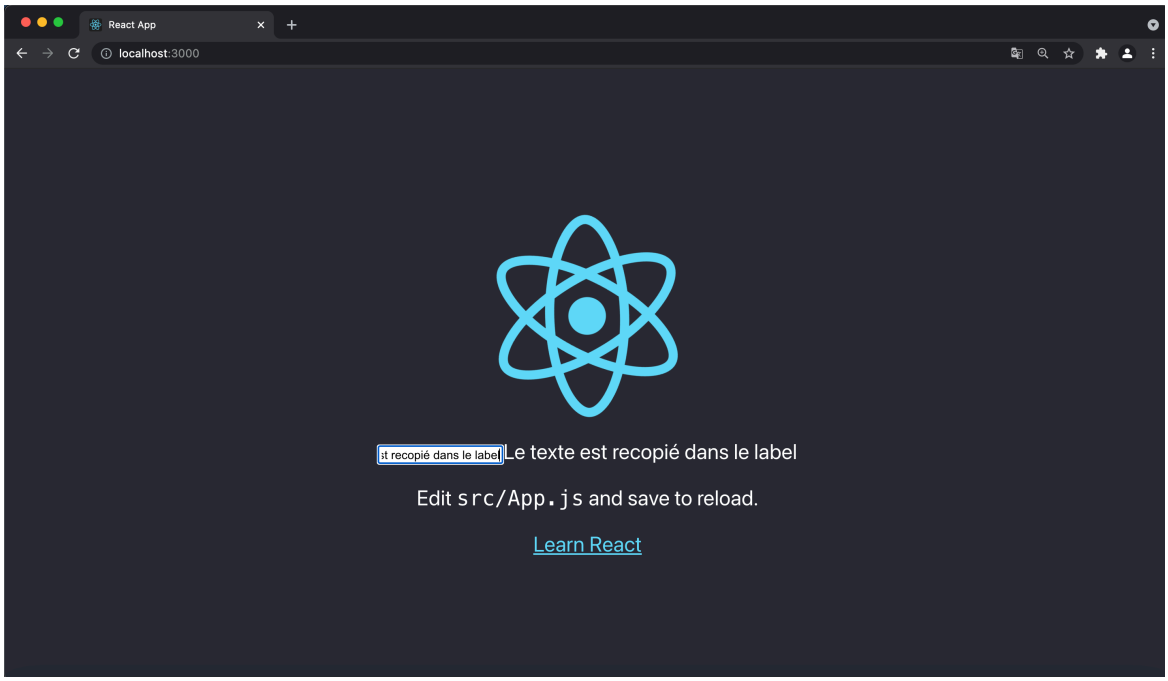
}

Ligne 11 nous faisons un bind de la fonction textChange ligne 14.
« Le bind() » est une méthode intégrée dans React qui est utilisée pour transmettre les données en tant qu'argument à la fonction d'un composant basé sur une classe. »

De la ligne 18 à 27 nous réalisons l'envoi de notre pattern. Nous attachons la fonction textChange à l'attribut onChange de la balise input.

Et nous affichons nomAttribut dans la balise label.
Pour finir la commande ‘exporter default’ est nécessaire afin d’ouvrir notre composant à l’export.

Voici le rendu :



Composant Fonctionnel

Nous allons reproduire le même composant que précédemment mais avec cette fois une fonction.

```
1  import { useState } from "react"  4.1k (gzipped: 1.8k)
2
3  function MonComposantF(){
4
5      const [textInput, setTextInput] = useState('');
6
7
8      return(
9          <div>
10         <input type="text" onChange={(e)=>{setTextInput(e.target.value)}}/>
11
12         <label>{textInput}</label>
13         </div>
14     )
15
16 }
17
18 export default MonComposantF;
```

Dans l'exemple ci-dessus nous avons remplacé le state par un Hook d'état
const [textInput, setTextInput] = useState('');

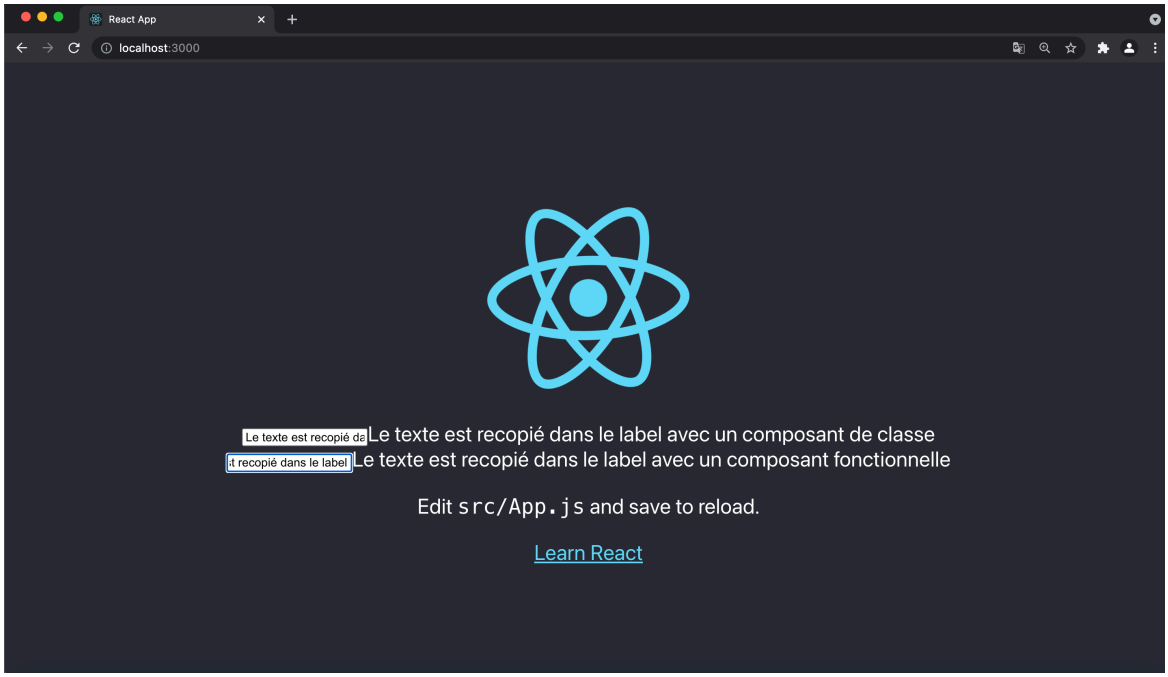
Ensuite nous déclarons une fonction anonyme dans l'attribut onChange sous forme de fonction fléché.

<input type="text" onChange={(e)=>{setTextInput(e.target.value)}}/>

Et ligne 12 nous faisons appel à notre Hook pour l'affichage du texte dans le label.

<label>{textInput}</label>

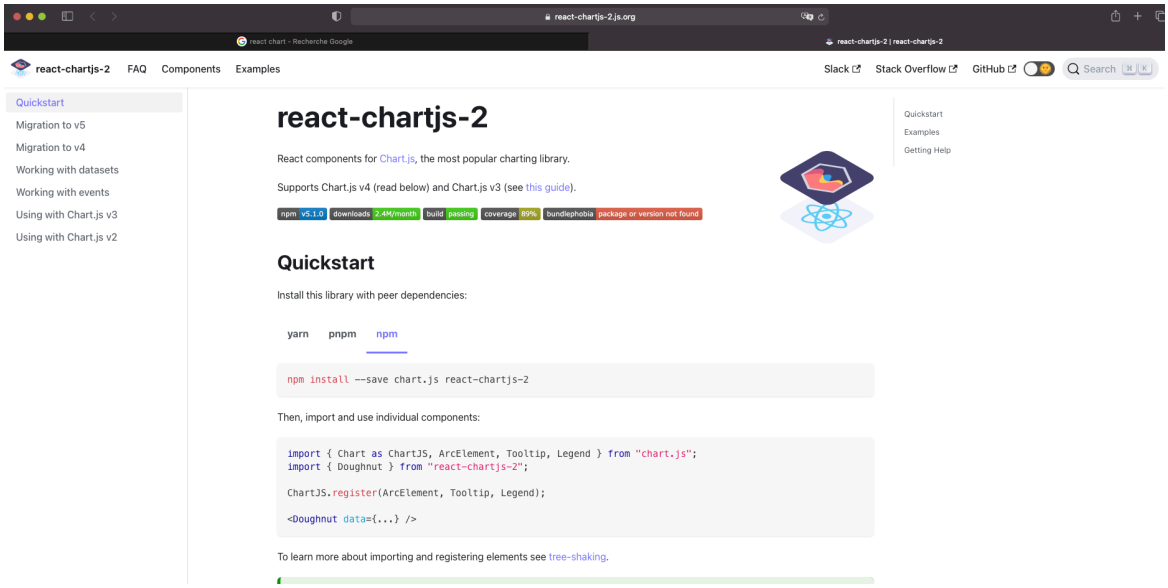
-Les Hooks sont des fonctions qui permettent de « se brancher » sur la gestion d'état local et de cycle de vie de React depuis des fonctions composants. Les Hooks ne fonctionnent pas dans des classes : ils vous permettent d'utiliser React sans classes. (Nous ne recommandons pas de ré-écrire vos composants existants du jour au lendemain, mais vous pouvez si vous le souhaitez commencer à utiliser les Hooks dans vos nouveaux composants.)



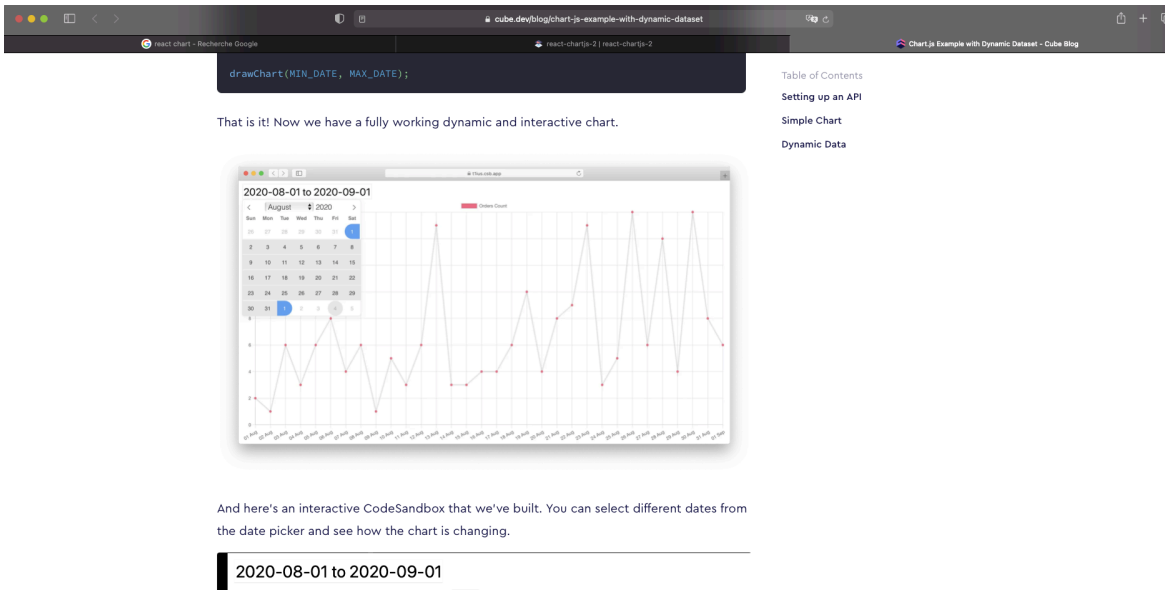
Voici le rendu avec le composant fonctionnel.

Npm install module

Plusieurs librairies existent et nous facilite la manipulation de différents modules. En effet nous pouvons installer des modules de thèmes bien précis de toute sorte. Il suffit d'intégrer la librairie associée à notre projet pour pouvoir l'utiliser.



Commandes d’installation de la librairie react-chartjs-2.



Voici un exemple du rendu des fonctions de cette librairie.

Pour l'installation de cette librairie il faudra taper dans le terminal à la racine du projet la commande :

npm install --save chart.js react-chartjs-2

Pour la désinstaller il suffira de taper :

npm uninstall --save chart.js react-chartjs-2

Sources

React: -https://fr.reactjs.org/docs/getting-started.html

Rendu des éléments : -https://fr.reactjs.org/docs/rendering-elements.html

Composants et Props: -https://fr.reactjs.org/docs/components-and-props.html

Hooks: -https://fr.reactjs.org/docs/hooks-overview.html

React Sur Windows: -https://learn.microsoft.com/fr-fr/windows/dev-environment/javascript/react-beginners-tutorial

Démarrer avec React: -https://developer.mozilla.org/fr/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

CreateElement(): -https://beta.reactjs.org/reference/react/createElement

React-chartjs-2 :-https://react-chartjs-2.js.org