

I. Introduction

[React](#) est un framework JavaScript populaire pour créer des applications front-end. Créé à l'origine par Facebook, il a gagné en popularité en permettant aux développeurs de créer des applications rapides à l'aide d'un paradigme de programmation intuitif qui lie JavaScript à une syntaxe de type HTML connue sous le nom de **JSX**.

```
const element = <h1>Bonjour, monde !</h1>;
```

Le démarrage d'un nouveau projet React était auparavant un processus complexe en plusieurs étapes qui impliquait la mise en place d'un système de construction, un [transcompilateur](#) de code pour convertir la syntaxe moderne en code lisible par tous les navigateurs et une structure de répertoire de base.

Mais maintenant, Create React App inclut tous les packages JavaScript dont vous avez besoin pour exécuter un projet React. Il comprend également un serveur avec rechargement à chaud qui actualisera votre page au fur et à mesure que vous apporterez des modifications au code. Enfin, il créera une structure pour vos répertoires et composants afin que vous puissiez vous lancer et commencer à coder en quelques minutes seulement.

React peut aussi être utilisé pour créer des applications mobiles grâce à [React Native](#)

Pour tester React, vous pouvez utiliser des bacs à sable en ligne comme [CodePen](#), [CodeSandbox](#), [Glitch](#) ou [Stackblitz](#), ou l'installer sur votre ordinateur.

React a besoin de node pour fonctionner. Nous allons donc commencer par nous familiariser avec Node

II. Node.js

[node.js](#) est un environnement d'exécution (*runtime environment*) construit sur l'interpréteur [JavaScript V8](#) de chrome.

- Il respecte les spécifications ECMAScript.
- Il permet d'exécuter du code *JavaScript*, mais aussi de bâtir facilement un serveur web, grâce à une bibliothèque intégrée : [HTTP](#).

Node.js, permet de créer des applications web avec des connexions bidirectionnelles où le côté serveur et le côté client peuvent communiquer en temps réel et échanger des données. En effet, Node.js a été révolutionnaire pour les développeurs qui voulaient pousser des applications web en temps réel sur WebSocket.

A. Installation

1. Les versions

Node.js propose deux versions différentes à télécharger : la version LTS et la version actuelle.

La version LTS (Long Term Support) ou support à long terme, indique la version qui est sur le marché depuis un certain temps et qui est fournie avec tout le support obligatoire. Par conséquent, l'on peut accéder à un grand nombre d'informations et de communautés pour une aide supplémentaire avec cette version.

Cette version LTS est recommandée à la plupart des utilisateurs en raison de sa durabilité et de son cycle de support de 18 mois. Comme il s’agit d’une version stable, son utilisation pour produire des backends peut aider à obtenir un résultat robuste.

La version actuelle (Current) indique la dernière version publiée de Node avec les caractéristiques les plus récemment ajoutées et mises à jour. Mais cette version a moins de support derrière elle (environ huit mois) et une possible exposition aux bogues. Par conséquent, les experts suggèrent d’utiliser cette version uniquement pour le développement frontend.




2. Procédure

- Rendez-vous sur le site <https://nodejs.org/en/download>

Downloads

Latest LTS Version: 20.10.0 (includes npm 10.2.3)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features																						
 Windows Installer <small>node-v20.10.0-x64.msi</small>	 macOS Installer <small>node-v20.10.0.pkg</small>	 Source Code <small>node-v20.10.0.tar.gz</small>																					
Windows Installer (.msi) Windows Binary (.zip) macOS Installer (.pkg) macOS Binary (.tar.gz) Linux Binaries (x64) Linux Binaries (ARM) Source Code	<table border="1"> <tr> <td>32-bit</td> <td>64-bit</td> <td>ARM64</td> </tr> <tr> <td>32-bit</td> <td>64-bit</td> <td>ARM64</td> </tr> <tr> <td colspan="3">64-bit / ARM64</td> </tr> <tr> <td>64-bit</td> <td colspan="2">ARM64</td> </tr> <tr> <td colspan="3">64-bit</td> </tr> <tr> <td>ARMv7</td> <td colspan="2">ARMv8</td> </tr> <tr> <td colspan="3">node-v20.10.0.tar.gz</td> </tr> </table>		32-bit	64-bit	ARM64	32-bit	64-bit	ARM64	64-bit / ARM64			64-bit	ARM64		64-bit			ARMv7	ARMv8		node-v20.10.0.tar.gz		
32-bit	64-bit	ARM64																					
32-bit	64-bit	ARM64																					
64-bit / ARM64																							
64-bit	ARM64																						
64-bit																							
ARMv7	ARMv8																						
node-v20.10.0.tar.gz																							

- Téléchargez la version Current “21,x”, correspondant à votre système d’exploitation
- Démarrez l’installation avec les options par défaut (gestionnaire de paquets NPM)

3. Vérification

- Pour vérifier l’installation placez-vous dans un terminal ou invite de commande
- Tapez les commandes suivantes

node --version pour connaître la version de node.js

npm --version pour connaître la version de npm

B. NPM

Npm est un gestionnaire de paquet. Il gère les dépendances des paquets.

La mise à jour régulière de npm permet également de mettre à jour vos paquets locaux et d’améliorer le code utilisé dans vos projets. Cependant, comme npm s’installe automatiquement avec la version de Node.js que vous choisissez, il manque souvent la dernière version de npm. Dans ce cas, vous pouvez vérifier votre version de npm et la mettre à jour manuellement en suivant un processus simple.

Les processus de vérification et de mise à jour de votre version de npm sont très similaires entre Windows, macOS et Linux – vous lancerez la même commande sur chacun d’eux.

`npm --version`

Pour mettre à jour la version, il suffit de lancer la commande

`npm install -g npm@latest`

III. Premier projet React

Ouvrez un terminal à l'emplacement de vos différents projets

- Tapez la commande suivante

```
npx create-react-app react-starter
```

Success! Created react-starter at /react-starter

Inside that directory, you can run several commands:

`npm start`

Starts the development server.

`npm run build`

Bundles the app into static files for production.

`npm test`

Starts the test runner.

`npm run eject`

Removes this tool and copies build dependencies, configuration files and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

`cd react-starter`

`npm start`

Happy hacking!

Ceci va créer une application react dans un dossier « react-starter »

Nb : create-react-app ne gère pas la logique backend ou les bases de données. Vous pouvez l'utiliser avec n'importe quel backend. Lorsque vous créez un projet, vous obtenez un dossier contenant du HTML statique, du CSS et du JS

- Placez-vous dans le dossier « react-starter »
- Pour démarrer votre application react en mode dev, tapez la commande

```
npm start
```

Vous avez un serveur node

Compiled successfully!

You can now view **react-starter** in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.1.61:3000

Note that the development build is not optimized.

To create a production build, use `npm run build`.

webpack compiled **successfully**



R4.10 : Complément web

TP #2 : React JS /1

- Votre navigateur ouvre automatiquement à l'adresse <http://localhost:3000>



- Ouvrez le projet dans vscode

	<p>Nous arrivons dans un environnement node Le dossier “node_modules” contient l’ensemble des packages (modules ,dépendances) nécessaires à l’application</p> <p>Le fichier “package.json” contient la liste des dépendances installées</p> <p>Le dossier “public” contient un fichier index.html où notre code sera injecté et servi au navigateur.</p> <p>Dans le dossier ”src”, vous trouverez le code de l’application, c’est ici en général que vous travaillerez</p>
--	--

Nous pouvons maintenant personnaliser notre application, en modifiant le fichier « 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;
```

Observez la syntaxe proche du HTML, on ne s'y trompe pas, car l'attribut « class » en HTML devient **className**, de même la source de l'image est une variable : `src={variable js}`, le JSX mélange les syntaxe HTML et JS.

La fonction App est ce qu'on appelle un **composant** dans React

- Modifiez le fichier comme suit

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

function App() {
  return (
    <div className="App">
      <header className="App-header">Bonjour React</header>
    </div>
  );
}

export default App;
```

Dès que vous enregistrez une page, le navigateur se rafraichit :



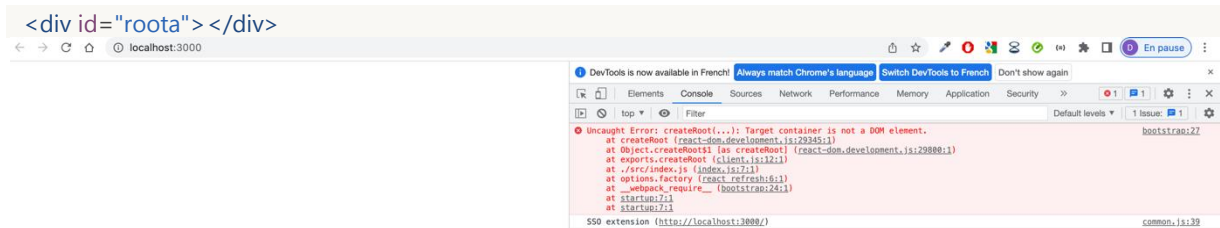
Bonjour React

Pour modifier le titre de la fenêtre dans le navigateur, il faudra modifier la balise « title » dans le fichier « **public/index.html** »

```
-->
<title> Mon premier projet React</title>
</head>
<body>
  <noscript> You need to enable JavaScript to run this app.</noscript>
  <div id="root"> </div>
```

NB : ce fichier est très important, car c'est lui qui est servi au navigateur

- Modifiez-le comme suit :



Nous avons cassé l'application, car on a modifié l'identifiant de l'objet HTML que React utilise pour injecter ses composants (cf src/index.js)

- Annulez la modification pour faire fonctionner l'application

```
<div id="root"> </div>
```

NB: Le script principal utilisé par react est le fichier « **src/index.js** »

- Ouvrez le fichier « src/index.js »

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

C'est le point d'entrée de l'application, il comporte 3 parties :

- Les instructions `import` : les modules react, une feuille de style et le composant « App »

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
```

- Le traitement de l'objet « root », que nous avons vu précédemment.

```
const root = ReactDOM.createRoot(document.getElementById("root"));
```

- La dernière instruction, permet de préciser le composant à afficher ici « App », dans l'élément du DOM #root.

```
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Toutes les applications React utilisent, quasiment toujours, la même structure

NB : Remarque la Barre fermante de la balise <App>. En JSX, les composants React et les éléments HTML doivent avoir des barres obliques (slash) de fermeture. Écrire uniquement <App> ou simplement provoquera une erreur.

IV. Les composants

Les applications React sont la plupart du temps construites en respectant une logique de **composants**. On va, par exemple, d'abord créer **le header**, puis **la sidebar ou le footer**, ect... Et pour chacun de ces composants on va créer un **fichier** qui sera importé lors de la compilation de l'application par webpack . Cela nous évite de devoir réécrire plusieurs fois le même block de code et nous permet de garder notre dossier de projet propre (pas de fichier de 3000 lignes). Les composants vont pouvoir être créés de **deux manières**: en créant une **fonction** ou en créant une **classe**.

En résumé : Les composants permettent de découper l'interface utilisateur en éléments indépendants et réutilisables, vous permettant ainsi de considérer chaque élément de manière isolée.

La création de composants-fonction React est relativement simple à prendre en main, aussi simple qu'une fonction:

```
// On crée le composant
function MonComposant() {
  return <div>Ceci est mon composant!</div> // Element visible
}
```

```
// On peut aussi créer un composant comme une fonction fléchée
const MonComposant = () => {
  return <div>Ceci est mon composant!</div> // Element visible
}
```

Prenez l'habitude de nommer les composants en mettant la première lettre en Majuscule

NB: Un composant doit obligatoirement renvoyer un flux HTML et un seul.

```
function MonComposant() {
  return (
    <div>1er élément</div>
    <div>2 ème élément</div>
  );
}
```

Les expressions JSX doivent avoir un élément parent.
Parsing error: Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>? (4:6)

Si vous souhaitez retourner plusieurs éléments il faudra les inclure dans une balise parente

```
function MonComposant() {
  return (
    <>
      <div>1er élément</div>
      <div>2 ème élément</div>
    </>
  );
}
```

NB: Pour rendre disponible votre composant aux autres composants il faudra l'exporter

```
export default MonComposant;
```

Et voilà ! On a un composant utilisable dans notre application React. En effet, un composant React fonctionnel est tout simplement une fonction qui va retourner un bloc de JSX.

1. Extensions Vscod

De nombreuses extensions de Vscod nous apportent des facilités

- Pour créer rapidement le squelette d'un composant
 - Installez les extension ES7+ React/Redux/React-Native snippets v4.4.3
 - Placez-vous sur une ligne de code vide, dans votre fichier jsx
 - Essayez les différentes commandes

Commande	Résultat
rafce	<pre>import React from 'react' export const Composant = () => { return (<div>Composant</div>) }</pre>
rafce	<pre>import React from 'react' const Composant = () => { return (<div>Composant</div>) } export default Composant</pre>
rfce	<pre>import React from 'react' export default function Composant() { return (<div>Composant</div>) }</pre>
rfce	<pre>import React from 'react' function Composant() { return (<div>Composant</div>) } export default Composant</pre>

- Pour importer automatiquement la déclaration d'un composant lors de son utilisation
- Installez Auto Importv1.5.4 de steotates



R4.10 : Complément web

TP #2 : React JS /1

B. Notre premier composant

- Dans le dossier src, créez un sous-dossier « composants »
- Créez à l'intérieur un fichier « Formulaire.jsx »
- Placez-vous dans le fichier vide et tapez ra et sélectionnez rafce

```
import React from 'react'

const Formulaire = () => {
  return (
    <div>Formulaire</div>
  )
}

export default Formulaire
```

VsCode vous a généré automatiquement un composant qui correspond au nom de votre fichier

Nb: Grace à la commande **export default** votre composant est désormais utilisable

Nous allons donc l'utiliser dans le composant principal du projet : "App"

- Ouvrez le fichier App.jsx et importez-y le composant Formulaire

```
import './App.css';
import Formulaire from './components/Formulaire'; // Import du composant formulaire

function App() {
  return (
    <div className="m-5">
      <header className="title">
        <h1 className="text-center">Mon formulaire</h1>
      </header>

      <div className="center">
        /* Utilisation du composant */
        <Formulaire />
      </div>
    </div>
  );
}

export default App;
```

Ce qui doit produire l'affichage suivant

Mon formulaire

Formulaire

- Modifiez votre formulaire en y ajoutant 3 zones de texte et un bouton

Mon formulaire

Nom :

Prénom :

Mail :

1. Mise en forme avec Bootstrap

Nous allons améliorer l’affichage du formulaire en utilisant les classes css de Bootstrap

```
import React from "react";
```

```
const Formulaire = () => {  
  return (  
    <div className="container py-4 px-3 mx-auto">  
      <form>  
        <div className="mb-3">  
          <label htmlFor="txtName">Nom :</label>  
          <input id="txtName" name="name" className="form-control" />  
        </div>  
        <div className="mb-3">  
          <label htmlFor="txtFirstName">Prénom :</label>  
          <input id="txtFistname" name="firstname" className="form-control" />  
        </div>  
        <div className="mb-3">  
          <label htmlFor="txtMail">Mail :</label>  
          <input id="txtMail" name="mail" className="form-control" />  
        </div>  
        <div className="mt-3 text-end">  
          <button type="submit" className="btn btn-primary">  
            Valider  
          </button>  
        </div>  
      </form>  
    </div>  
  );  
};
```

```
export default Formulaire;
```

Notez qu’avec React l’attribut class devient className comme en JS, pour voir les autres attributs qui changent de notation, rendez-vous sur le site de [react](https://react.dev)

Il ne nous reste qu’à installer Bootstrap en tant que dépendance

- Arrêtez le serveur “CTRL+C”
- Installez bootstrap

```
npm install bootstrap
```

- Relancez le serveur

```
npm start
```

- Importez le css de bootstrap dans notre composant App.js

```
import "bootstrap/dist/css/bootstrap.css"
```



R4.10 : Complément web

TP #2 : React JS /1

Mon formulaire

Nom :

Prénom :

Mail :

Valider

C. Passage de paramètres

1. Props

Les **props** sont les entrées d'un composant React. Elles sont passées d'un composant parent à un composant enfant. Elles sont équivalentes aux paramètres de fonction

NB : Les props sont en lecture seule. Elles ne doivent jamais être modifiées.

Dans l'exercice précédent, vous avez créé un composant Formulaire qui contient des éléments input et un bouton

```
const Formulaire = () => {  
  return (  
    <div className="container py-4 px-3 mx-auto">  
      <form>  
        <div className="mb-3">  
          <label htmlFor="txtName">Nom :</label>  
          <input id="txtName" name="name" className="form-control" />  
        </div>  
        <div className="mb-3">  
          <label htmlFor="txtFirstName">Prénom :</label>  
          <input id="txtFistname" name="firstname" className="form-control" />  
        </div>  
        <div className="mb-3">  
          <label htmlFor="txtMail">Mail :</label>  
          <input id="txtMail" name="mail" className="form-control" />  
        </div>  
        <div className="mt-3 text-end">  
          <button type="submit" className="btn btn-primary">  
            Valider  
          </button>  
        </div>  
      </form>  
    </div>  
  );  
};  
export default Formulaire;
```

Sachant que les composants doivent permettre de simplifier les interfaces et être réutilisables, on pourrait décomposer notre composant « Formulaire » en sous-composant homogènes

Un premier niveau serait des blocs de ligne :

```
<div className="mb-3">  
  <label htmlFor="txtName">Nom :</label>  
  <input id="txtName" name="name" className="form-control" />  
</div>
```

- Dans le dossier « composants », créez un sous-composant FormBlockInput

```
const FormBlockInput = (props) => (  
  <div className="mb-3">  
    <label htmlFor={props.id}>{props.label}</label>  
    <input id={props.id} name={props.name} className="form-control" />  
  </div>);  
export default FormBlockInput;
```

NB : les attributs (id, name, label) de l'objet « props » seront passés par le composant parent, de la manière suivante

```
< FormBlockInput id="xxx" label="yyy name="zzz" />
```

NB : Tous les attributs fournis par le composant parent, sont récupérables dans l'objet props du fils.

Un attribut key= « xx », avec xx = un identifiant unique , doit être ajouté sur les composants multiples de même nature, par exemple les li dans un ul

- Créez un sous-composant FormBlockButton
- Modifier le composant Formulaire pour appeler les sous-composants FormBlockInput et FormBlockButton
- Vous pouvez encore décomposer le sous-composant FormBlockInput, par exemple pour appeler un sous-composant FormInput

*Pour voir la structure de vos composants et les déboguer , il faut installer l'extension **React devTools** dans votre navigateur.*

2. Props.children

props.children est disponible dans chaque composant. Elle référence le contenu présent entre les balises ouvrante et fermante du composant. Par exemple :

```
<Welcome>Bonjour monde !</Welcome>
```

Le texte **Bonjour monde !** est présent dans la **props.children** du composant **Welcome** :

```
function Welcome(props) {  
  return <p>{props.children}</p>;  
}
```



Fil Rouge #01