

## 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>

### Download Node.js®

Get Node.js®  for  using  with

```
1 # Download and install nvm:
2 curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
3
4 # Download and install Node.js:
5 nvm install 22
6
7 # Verify the Node.js version:
8 node -v # Should print "v22.13.1".
9 nvm current # Should print "v22.13.1".
10
11 # Verify npm version:
12 npm -v # Should print "10.9.2".
```

Bash

Copy to clipboard

"nvm" is a cross-platform Node.js version manager. If you encounter any issues please visit [nvm's website](#)

- Téléchargez la version Current, 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

## 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 connaître la version de npm

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

**npm install -g npm@latest**

### C. Vite

En complément, nous utiliserons le bundler « vite.js » pour coder du js moderne.

Un bundler, ou compilateur de modules, est un outil qui regroupe les fichiers JavaScript, CSS, HTML, images, polices, etc. et leurs dépendances en un ou plusieurs fichiers optimisés, prêts à être servis à un navigateur web.

Imaginez votre projet web comme une voiture :

- Vos fichiers de code (JS, CSS, HTML...) sont les pièces détachées.
- Le bundler est l'usine d'assemblage.
- Le fichier final optimisé est la voiture prête à rouler.

#### 1. Pourquoi utiliser un bundler ?

- Performance: Le bundler optimise vos fichiers pour une livraison plus rapide (minification, découpage du code, etc.).
- Compatibilité: Il gère les différentes versions de JavaScript et CSS pour une meilleure compatibilité avec les navigateurs.
- Modules: Il permet d'utiliser des modules JavaScript (comme ceux de npm) de manière organisée et efficace.
- Développement simplifié: Les bundlers offrent des fonctionnalités comme le rechargement à chaud (HMR) et la gestion des assets qui facilitent le développement.

#### 2. Comparaison avec Webpack et autres Bundlers:

Fonctionnalité	Vite	Webpack	Parcel
<b>Approche</b>	ES Modules natifs	Bundling	Bundling
<b>Vitesse dev</b>	🚀 Extrêmement rapide	Rapide	Moyenne
<b>Configuration</b>	Minimaliste	Avancée	Minimaliste
<b>Maturité</b>	Récent	Mature	Récent

### III. Premier projet React

Ouvrez un terminal à l'emplacement de vos différents projets, par exemple « R4.10/TP/React »

- Tapez la commande suivante

```
npm create vite@latest react-starter
```

```
> npx
> create-vite react-starter

? Select a framework: > - Use arrow-keys. Return to submit.
  Vanilla
  Vue
> React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Angular
  Others

✓ Select a framework: > React
? Select a variant: > - Use arrow-keys. Return to submit.
  TypeScript
  TypeScript + SWC
> JavaScript
  JavaScript + SWC
  React Router v7 ↵

Done. Now run:

  cd react-starter
  npm install
  npm run dev
```

- Sélectionnez « React » comme framework

- A ce stade, sélectionnez « Javascript » qui utilisera babel comme transpiler

- Entrez les commandes précisées pour installer les modules et lancer le serveur de dev

Ceci va créer une application vite avec l'extension react dans le dossier « react-starter »

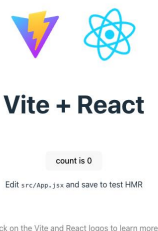
*Nb : react ne gère pas la logique backend ou les bases de données. Vous pouvez l'utiliser avec n'importe quel backend. Lorsque vous créez un projet, vous obtenez un dossier contenant du HTML statique, du CSS et du JS*

```
VITE v6.0.11 ready in 405 ms

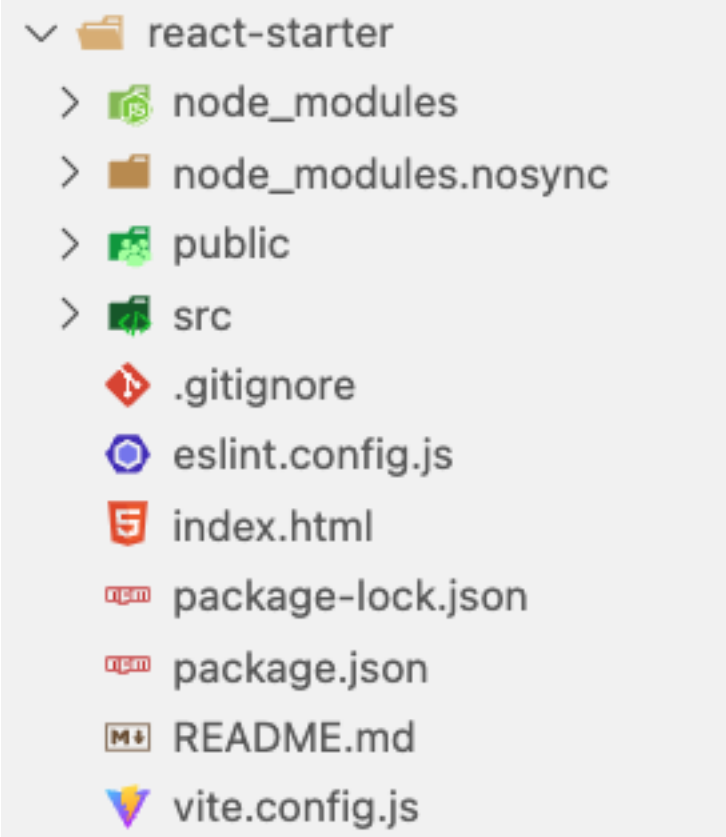
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

- Cliquez sur le lien en maintenant la touche option du clavier ou rendez-vous dans le navigateur à l'adresse indiquée

Vous avez un site react fonctionnel



- Ouvrez le projet dans vscode

 <p>react-starter</p> <ul style="list-style-type: none"> <li>node_modules</li> <li>node_modules.nosync</li> <li>public</li> <li>src</li> <li>.gitignore</li> <li>eslint.config.js</li> <li>index.html</li> <li>package-lock.json</li> <li>package.json</li> <li>README.md</li> <li>vite.config.js</li> </ul>	<p>Nous arrivons dans un environnement node</p> <p>Le dossier "node_modules" contient l'ensemble des packages (modules ,dépendances) nécessaires à l'application</p> <p>Le dossier "public" contient les ressources publiques, pour l'instant juste le logo de vite</p> <p>Dans le dossier "src", vous trouverez le code de l'application, c'est ici en général que vous travaillerez</p> <p>Le fichier « index.html » qui est affiché dans le navigateur</p> <p>Le fichier "package.json" contient la liste des dépendances installées</p> <p>Le fichier « vite.config.js » contient la configuration de vite, les extensions chargées par vite</p>
--	--

#### A. Le fichier index.html

```

<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>

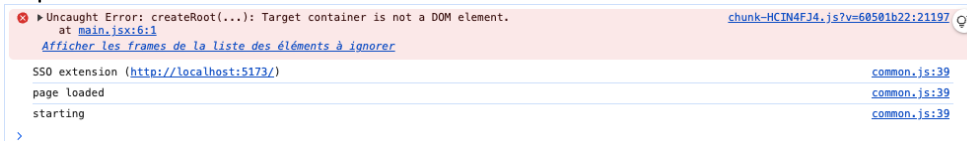
```

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

- Modifiez-le comme suit :

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

Votre page est cassée , Allez dans l'inspecteur de code et l'onglet « console », pour analyser ce qui se passe



Nous avons changé l'id , de l'élément central de la page , qui servait à l'affichage du composant App appelé dans le fichier « src/main.jsx »

Revenez à l'id précédent, pour corriger l'anomalie

## B. Le fichier src/main.jsx

02-React > react-starter > src > JS main.jsx

```
1 import { StrictMode } from 'react'
2 import { createRoot } from 'react-dom/client'
3 import './index.css'
4 import App from './App.jsx'
5
6 createRoot(document.getElementById('root')).re
7   <StrictMode>
8     | <App />
9   </StrictMode>,
10 )
11
```

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 »
- Le traitement de l'objet « root », que nous avons vu précédemment.
- La dernière instruction, permet de préciser le composant à afficher ici « App », dans l'élément du DOM #root.

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

*NB : Remarquez 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 <img> provoquera une erreur.*

## C. Le fichier src/App.jsx

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from './vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <div>
      <a href="https://vite.dev" target="_blank">
        <img src={viteLogo} className="logo" alt="Vite logo" />
      </a>
      <a href="https://react.dev" target="_blank">
        <img src={reactLogo} className="logo react" alt="React logo" />
      </a>
    </div>
    <h1>Vite + React</h1>
    <div className="card">
      <button onClick={() => setCount((count) => count + 1)}>
        count is {count}
      </button>
      <p>
        Edit <code>src/App.jsx</code> and save to test HMR
      </p>
    </div>
    <p className="read-the-docs">
      Click on the Vite and React logos to learn more
    </p>
  </div>
)
}

export default App
```

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

C'est le composant principal, appelé dans le main.jsx

Il contient la logique et les balises visibles affichées sur la page d'accueil



**Vite + React**

count is 0  
Edit src/App.jsx and save to test HMR

Click on the Vite and React logos to learn more

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 .

Nous allons maintenant personnaliser notre application, en modifiant le composant « App.jsx »

- Modifiez le fichier comme suit

```
import './App.css';
```

```
function App() {  
  return (  
    <>  
    <h1>Bonjour React</h1>  
    </>  
  );  
}
```

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

## Bonjour React

```
export default App;
```

- Modifiez le titre de la fenêtre dans le navigateur, dans le fichier « **index.html** »

## 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**, etc.... Et pour chacun de ces composants on va créer un **fichier** qui sera importé lors de la compilation de l'application par le bundler. 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>2ème 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>2ème 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 Vscode

De nombreuses extensions de Vscode nous apportent des facilités

## 1. 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
rafc	<pre>import React from 'react'  export const Composant = () =&gt; {   return (     &lt;div&gt;Composant&lt;/div&gt;   ) }</pre>
rafce	<pre>import React from 'react'  const Composant = () =&gt; {   return (     &lt;div&gt;Composant&lt;/div&gt;   ) }  export default Composant</pre>
rfc	<pre>import React from 'react'  export default function Composant() {   return (     &lt;div&gt;Composant&lt;/div&gt;   ) }</pre>
rfce	<pre>import React from 'react'  function Composant() {   return (     &lt;div&gt;Composant&lt;/div&gt;   ) }  export default Composant</pre>

- Pour importer automatiquement la déclaration d'un composant lors de son utilisation, installez [Auto Import](#) v1.5.4 de steotates



## 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 le composant « Formulaire » en y ajoutant un formulaire comportant 3 zones de texte et un bouton

```
import React from "react";
// "React" is defined but never used.

const Formulaire = () => {
  return (
    <div>
      <form>
        <div>
          <label htmlFor="username">Nom d'utilisateur</label>
          <input type="text" id="username" name="username" required placeholder="Entrez votre nom d'utilisateur" />
        </div>
        <label htmlFor="email">Email</label>
        <input type="email" id="email" name="email" required placeholder="Entrez votre email" />
        </div>
        <div>
          <label htmlFor="password">Mot de passe</label>
          <input type="password" id="password" name="password" required placeholder="Entrez votre mot de passe"/>
        </div>
        <div>
          <button type="submit">Valider</button>
        </div>
      </form>
    </div>
  );
};

export default Formulaire;
```

## Mon formulaire

Nom d'utilisateur

Email

Mot de passe

### 1. Mise en forme avec Tailwind CSS

Vite a déjà formaté notre affichage avec ses styles définis dans le fichier « src/index.css », il va falloir donc les surcharger dans notre configuration de Tailwind CSS

#### a) Installation de Tailwind CSS comme extension vite

- Exécutez la commande suivante dans un terminal à partir de la racine du projet

```
npm install tailwindcss @tailwindcss/vite
```

- Modifiez le fichier de configuration de vite, « vite.config.js » comme suit :

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'

// https://vite.dev/config/
export default defineConfig({
  plugins: [react(), tailwindcss()],
})
```

- Au début du fichier « src/App.css », ajoutez la ligne

```
@import "tailwindcss";
```

Ainsi Tailwind est intégré à votre projet, observez les changements dans le visuel du formulaire

## Mon formulaire

Nom d'utilisateur Entrez votre nom d'utilis

Email Entrez votre email

Mot de passe Entrez votre mot de pas:

Valider

C'est mieux mais pas suffisant.

Modifiez le fichier « src/App.css », en utilisant les directives Tailwind pour transformer le formulaire, en le centrant horizontalement etc...

## Mon formulaire

Nom d'utilisateur

Email

Mot de passe

Valider

Exemple de classes utilisées

- bg-white, p-8, rounded-lg, shadow-md** : pour le formulaire.
- mb-4** : entre les éléments du formulaire.
- border, border-gray-300, rounded-lg, p-2** : pour les champs de texte.
- focus:outline-none, focus:ring-2, focus:ring-blue-500** : focus sur les champs.
- w-full** : champs de saisie
- hover:bg-blue-600** : Change la couleur de fond du bouton lors du survol.

Je vous laisse voir comment éviter la bordure bleue qui s'affiche au survol du bouton ☺

*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)*

### 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.

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>
  <label htmlFor="email">Email</label>
  <input type="email" id="email" name="email" required placeholder="Entrez votre email" />
</div>
```

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

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

```
const FormBlockInput = (props) => {
  return (
    <div>
      <label htmlFor={props.id}>{props.label}</label>
      <input
        type="text"
        id={props.id}
        name={props.name}
        placeholder={props.placeholder}
      />
    </div>
  );
};
```

```
export default FormBlockInput;
```

```
<FormBlockInput id="username" label="Nom d'utilisateur" placeholder="Entrez votre nom d'utilisateur"></FormBlockInput>
```

NB : les attributs (id, name, label, placeholder) 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 composant FormBlockButton pour gérer le bouton
- Modifier le composant Formulaire pour appeler les 2 nouveaux composants
- Décomposez 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 #02

