



INSA

INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
HAUTS-DE-FRANCE

 **Université
Polytechnique**
HAUTS-DE-FRANCE

REACT.JS

Présentation générale

- <https://nodejs.org/fr/download>

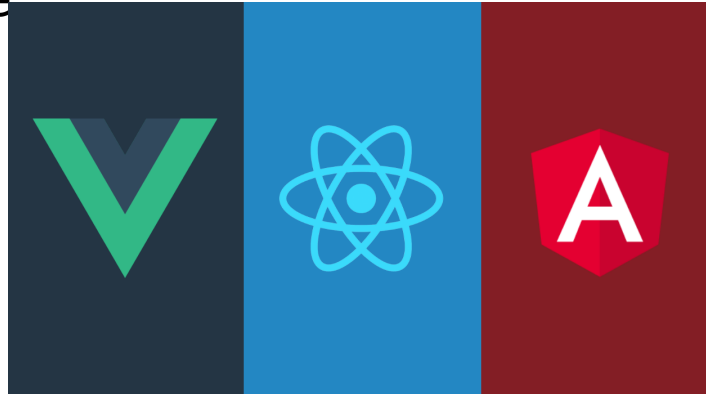
React - Définition et fonctionnalités

- React : Un framework JavaScript open-source développé par Facebook
- Utilisé pour la création d'interfaces utilisateur interactives et dynamiques
- Se concentre sur la construction de composants réutilisables pour une modularité maximale
- Utilise un Virtual DOM pour des mises à jour d'interface utilisateur efficaces et rapides



Utilité des frameworks pour les développeurs

- Un framework fournit des bibliothèques, des outils et des conventions qui simplifient le développement
- Permet aux développeurs de se concentrer sur la logique métier en résolvant des problèmes techniques courants
- Offre des avantages tels que la réutilisation du code, la modularité et la collaboration facilitée
- Suit des conventions établies pour faciliter la maintenance et la compréhension du code



R. Tomczak

React vs vue.js vs Angular

- **React, Vue.js et Angular** sont tous trois des frameworks ou des bibliothèques JavaScript populaires utilisés pour créer des applications web modernes.
- Ils ont chacun leurs forces et leurs particularités, voici une comparaison simple entre eux :
- **React (Bibliothèque JavaScript développée par Facebook)**
 - Favorise un modèle de programmation plus flexible et moins prescriptif.
 - L'accent est mis sur la création de composants réutilisables.
 - La bibliothèque elle-même est plus petite et plus légère que Angular.
 - Utilisé par Facebook, Instagram, Airbnb, et d'autres.

React vs vue.js vs Angular

■ **Vue.js (Framework JavaScript développé par Evan You, ancien employé de Google)**

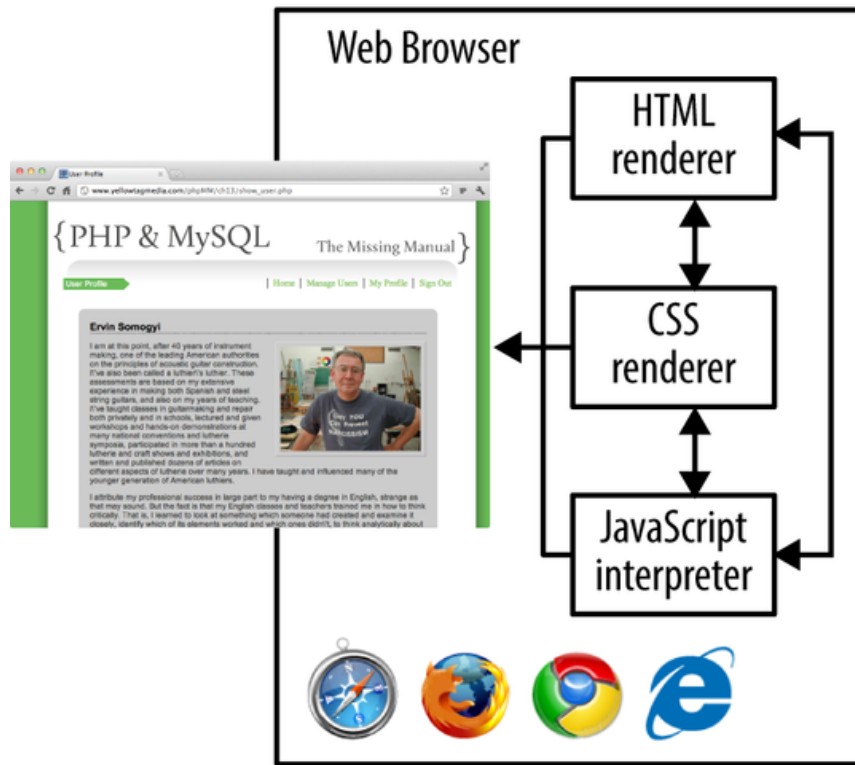
- Syntaxe simple et facile à comprendre, excellent pour les débutants
Se situe entre React et Angular en termes de flexibilité et de convention.
- Permet une intégration facile dans les projets existants.
- Utilisé par Alibaba, Xiaomi, et d'autres.

■ **Angular (Framework JavaScript développé par Google)**

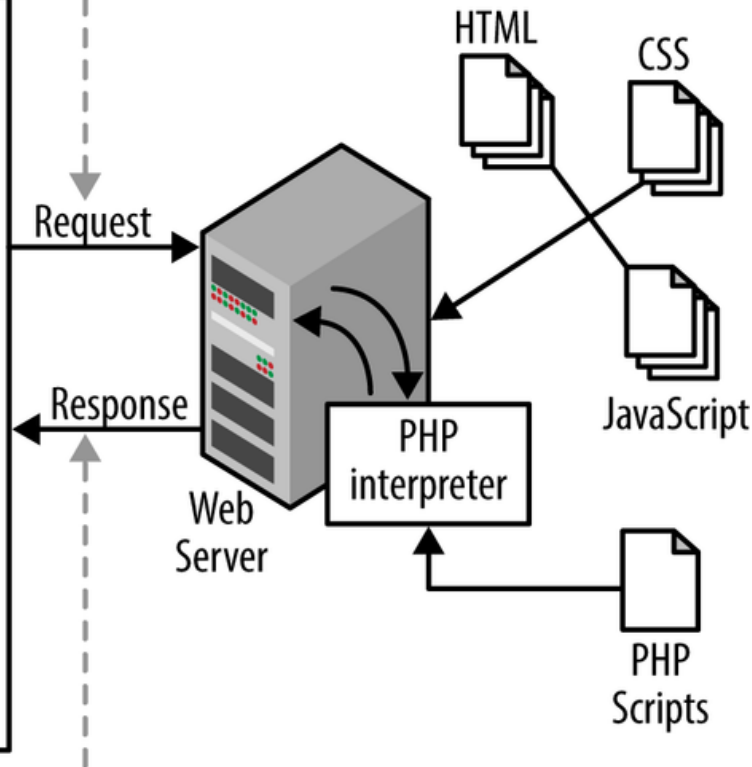
- Comprend un ensemble d'outils plus complet "out of the box" (gestion des formulaires, gestion de l'état, etc.).
- Suit une structure et des conventions strictes, ce qui peut faciliter la maintenance à grande échelle.
- Utilise le TypeScript pour le développement, ce qui peut améliorer la qualité du code et le débogage.
- Utilisé par Google, Microsoft, IBM, et d'autres

Tendance actuelle

- La tendance web actuelle sépare la partie client (HTML, CSS, JavaScript) et la partie serveur (PHP, Java) d'un site web.
- Traditionnellement, le site web envoie des pages HTML au navigateur à chaque requête, nécessitant des demandes fréquentes au serveur.



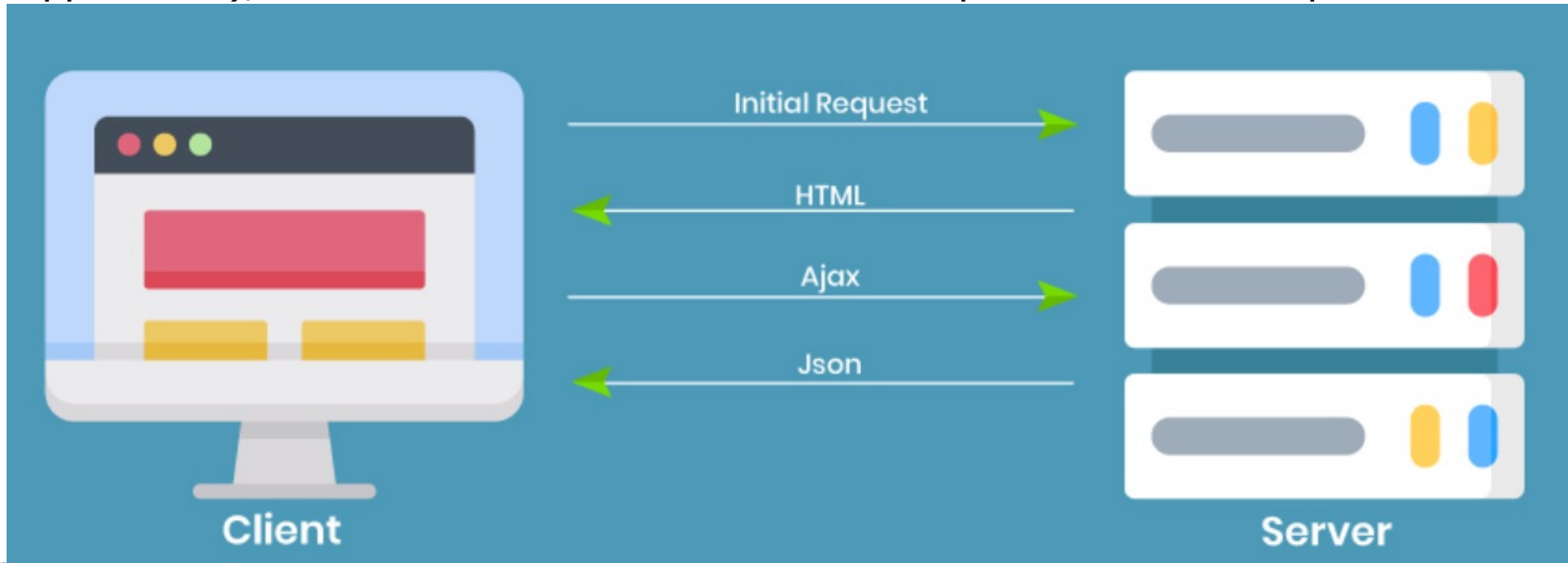
Could be for HTML, CSS,
PHP or a combination.



Response is not PHP,
but the result of
interpreting PHP, usually
more HTML and CSS.

Tendance actuelle

- Avec JavaScript, des applications web autonomes peuvent être créées, nécessitant un seul envoi de page par le serveur.
- Ce procédé offre une navigation plus fluide et rapide, sans rechargement de la page entière à chaque interaction.
- Cette méthode, connue sous le nom d'architecture ESPA (Single Page Applications), offre une meilleure réactivité et une expérience utilisateur plus



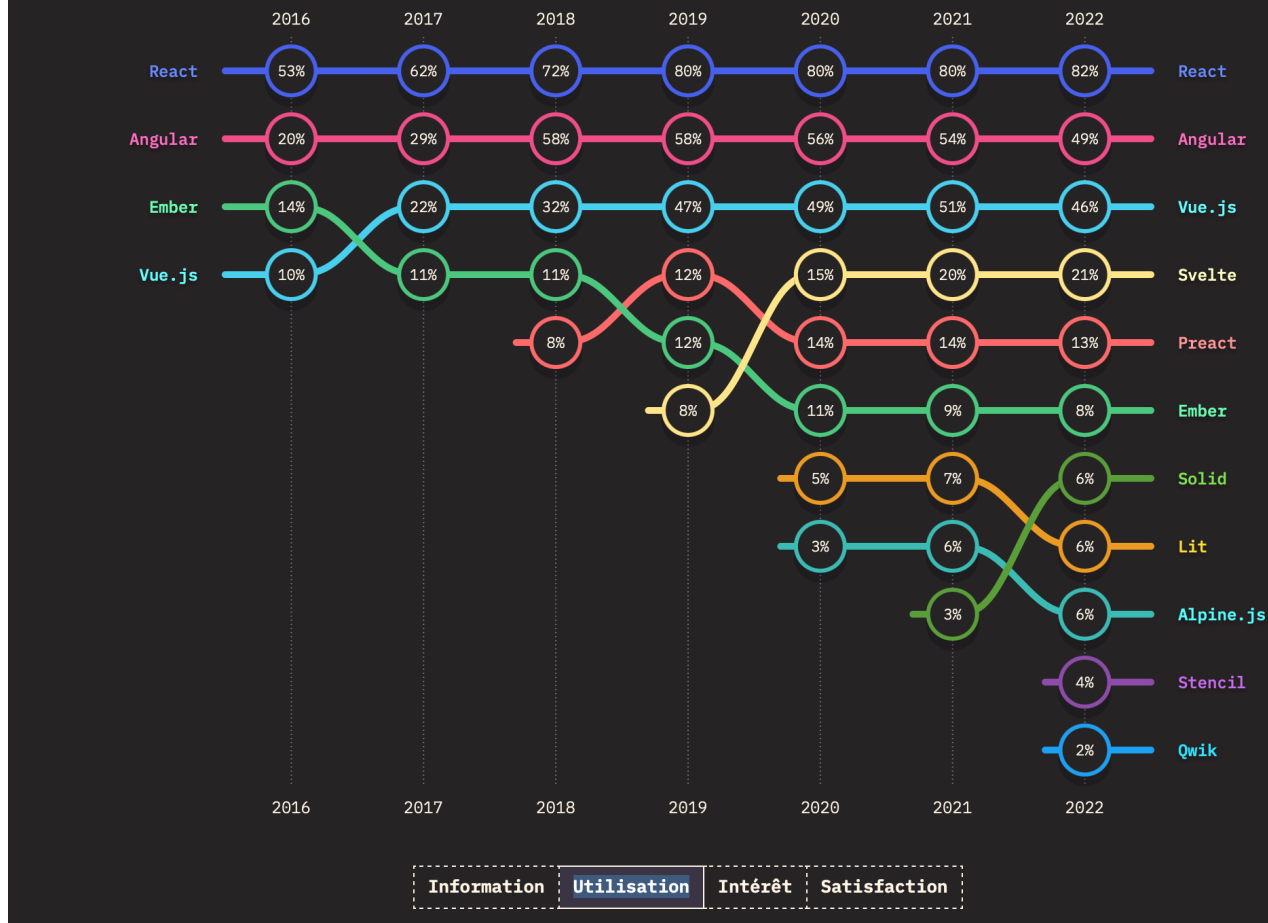
Présentation de React

Pourquoi choisir React?

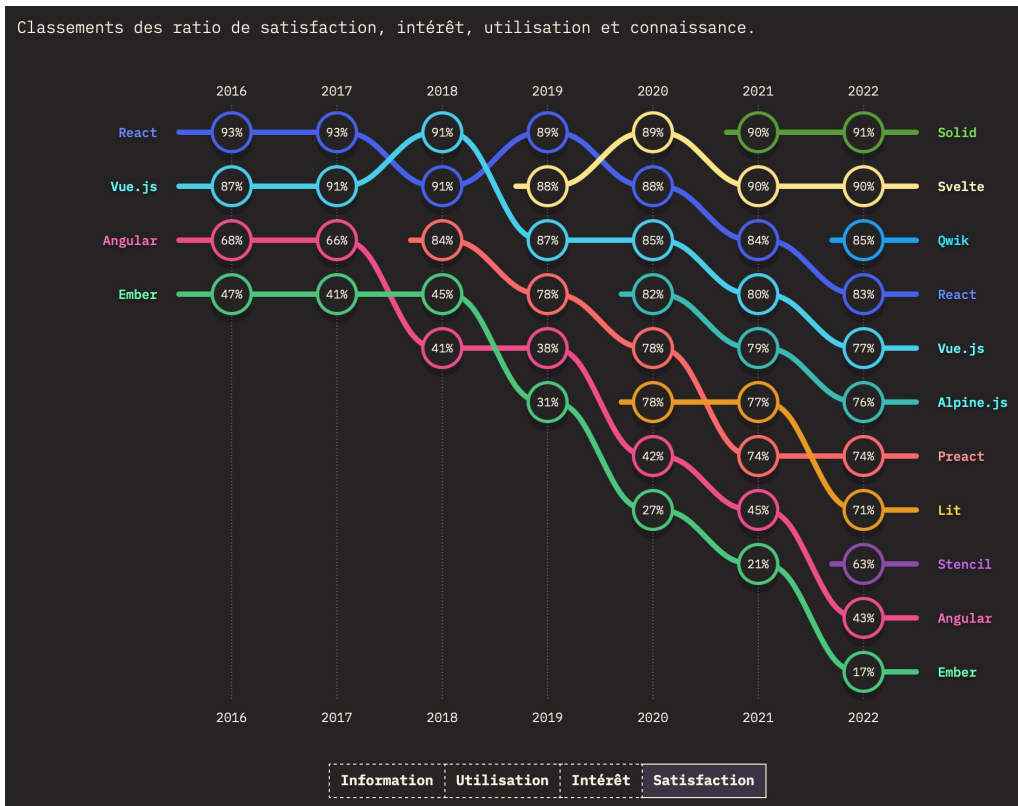
- Composants réutilisables
- Virtual DOM pour des performances optimisées
- Écosystème et communauté active
- Réactivité et expérience utilisateur fluide
- Support à long terme par Facebook
- Facilité d'apprentissage
- Créez des applications interactives et dynamiques
- Stabilité et évolution constante du framework

Utilisation

Classements des ratio de satisfaction, intérêt, utilisation et connaissance.

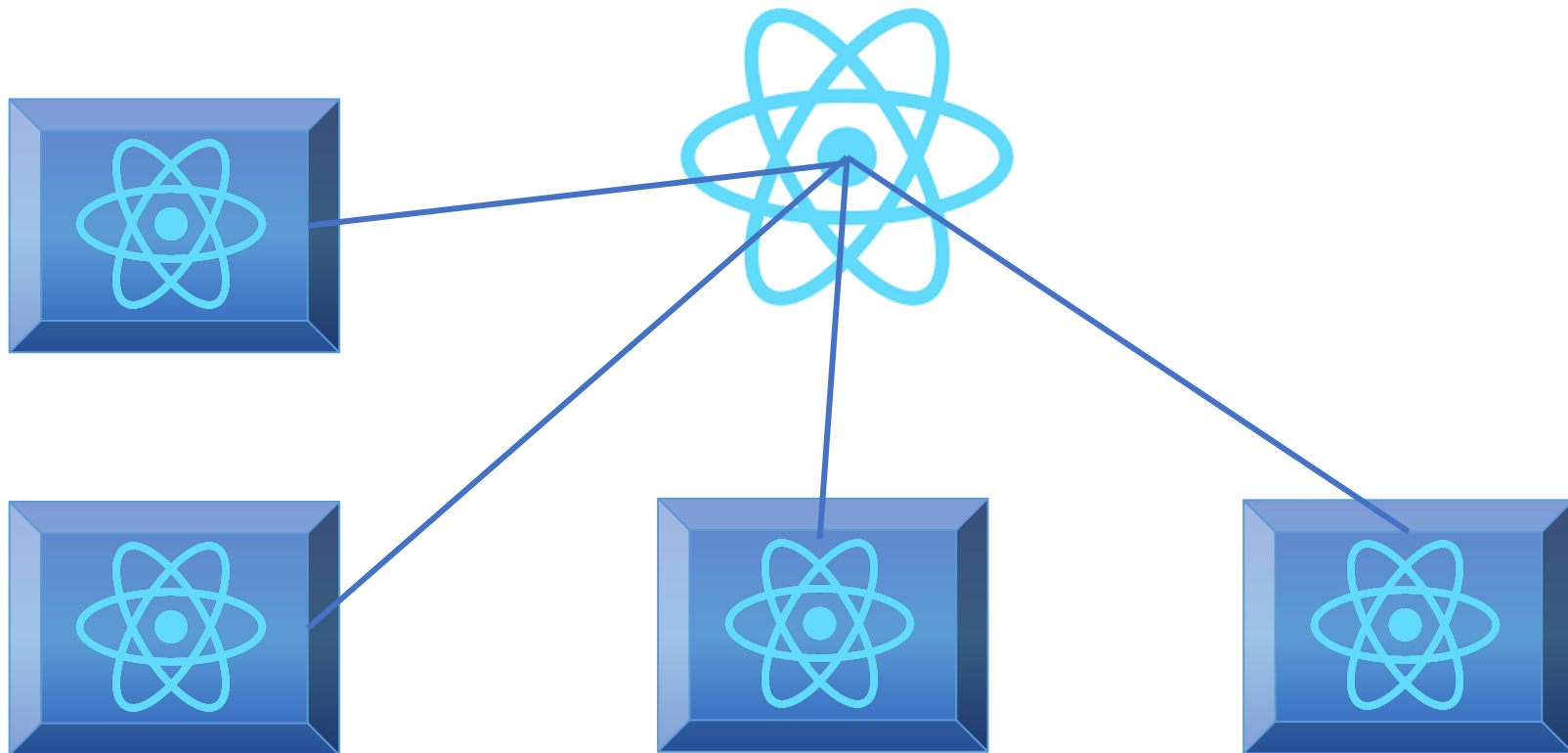


Satisfaction



Orienté composants

- React est un framework facilitant le développement d'applications grâce à une approche orientée composants.
- Cette approche consiste à créer des composants petits et indépendants formant ensemble une application complète.
- Chaque composant est autonome, incluant sa propre structure HTML, des règles CSS et une fonction JavaScript pour un comportement spécifique.
- Les composants React se basent sur le standard des Web Components, décomposant une page web selon ses différentes fonctionnalités.
- Ce standard n'est pas encore universellement supporté par les navigateurs, mais pourrait l'être à l'avenir.



React et ECMAScript 6

- Introduction à ECMAScript 6 (ES6) Publié en juin 2015
- Nouvelles fonctionnalités de ES6
 - Syntaxes et constructions plus concises et expressives
 - Classes pour la programmation orientée objet
 - Fonctions fléchées pour des fonctions anonymes plus concises
 - Modules pour la gestion des dépendances et de l'organisation du code
- Améliorations de la manipulation des tableaux
 - Méthodes map, filter et reduce pour une manipulation plus efficace
 - Promesses pour une gestion plus propre des opérations asynchrones
- Gestion avancée des variables
 - Mots-clés let et const pour un meilleur contrôle de la portée
 - Déstructuration d'objets et de tableaux pour un accès simplifié aux valeurs

TypeScript

- TypeScript est un langage open source développé par Microsoft.
- Ajoute des fonctionnalités de typage statique optionnelles à JavaScript.
- Détecte et prévient les erreurs de typage avant l'exécution du code.
- Étend les fonctionnalités d'ES6 avec des déclarations de types, des interfaces, des classes, des modules, etc.
- Compilé en JavaScript standard pour l'exécution dans un navigateur ou un environnement JavaScript.

Et REACT ?

- React.js : Bibliothèque JavaScript les interfaces utilisateur interactives
- Détection des erreurs de typage : JavaScript est un langage à typage dynamique, ce qui rend difficile la détection des erreurs de typage avant l'exécution
- Avantages de TypeScript :
 - Typage statique : Ajout de fonctionnalités de typage statique à JavaScript pour une détection précoce des erreurs de typage
 - Meilleure vérification des types, documentation du code, autocomplétion et refactoring plus sûr
- Compatibilité avec ES6 : TypeScript offre une prise en charge native des fonctionnalités avancées d'ES6 et des versions ultérieures
- Utilisation de fonctionnalités avancées : Modules, classes, interfaces, etc.
- Amélioration de la productivité et de la qualité du code dans le développement React.js avec TypeScript



Premiers pas avec React

Processus de création d'une application React

- Différentes méthodes pour démarrer un projet React.
 - Présentation de la méthode à partir d'un dossier vide.
 - Ajout des fichiers de configuration dans le dossier racine.
 - Création d'un dossier nommé "React SuperHeros App" (nom personnalisable).
-
- Objectif : Développer une application de gestion de Super Héros.
 - Processus complet jusqu'à une application finale prête pour la production.
 - Compréhension globale du développement d'une application React.

Point d'étape

JavaScript - TypeScript

Avantages/Inconvénients

■ TypeScript :

• Avantages :

- **Sûreté des types** : TypeScript est un langage à typage statique, ce qui peut aider à prévenir certains types d'erreurs pendant le développement.
- **Meilleure autocomplétion et refactoring** : Les outils de développement peuvent utiliser les informations de type pour fournir une meilleure autocomplétion et des outils de refactoring plus robustes.
- **Documentation du code** : Les types peuvent servir de documentation pour le code, ce qui peut rendre le code plus facile à comprendre pour les nouveaux venus sur le projet.

• Inconvénients :

- **Courbe d'apprentissage** : TypeScript peut être plus difficile à apprendre pour les développeurs qui n'ont pas d'expérience avec les langages à typage statique.
- **Complexité supplémentaire** : TypeScript ajoute une étape de compilation supplémentaire et peut rendre le code plus complexe.

■ JavaScript :

• Avantages :

- **Simplicité** : JavaScript est plus simple à utiliser car il n'y a pas de système de types à gérer.
- **Popularité** : JavaScript est un langage plus populaire, donc il peut être plus facile de trouver des développeurs JavaScript.

• Inconvénients :

- **Moins sûr** : Sans le système de types de TypeScript, il peut être plus facile de commettre certaines erreurs.

Pourquoi Typescript a été préféré à Javascript dans ce projet ?

Langage pour l'avenir ?

- TypeScript gagne en popularité :
 - Selon le sondage annuel de Stack Overflow en 2020, TypeScript est maintenant l'un des langages de programmation les plus aimés et recherchés. Cependant, prévoir si TypeScript est "l'avenir" est un peu plus complexe.
 - Voici quelques points à considérer :
- 1. **Popularité croissante** : TypeScript a gagné en popularité parmi les développeurs JavaScript pour plusieurs raisons, notamment pour sa sécurité de type statique, ses outils de développement améliorés, et sa compatibilité avec les nouvelles fonctionnalités de JavaScript.

Pourquoi Typescript a été préféré à Javascript dans ce projet ?

Langage pour l'avenir ?

1. **Adoption par l'industrie :** De grandes entreprises, comme Microsoft (qui a développé TypeScript), Google, Airbnb, et Slack, ont adopté TypeScript pour certains de leurs projets.
2. En outre, des **projets open-source importants**, comme Angular et Deno, utilisent également TypeScript.
3. **Favorisé par certains cadres :** Certains cadres de développement populaires, comme Angular, favorisent ou encouragent l'utilisation de TypeScript. Cela pourrait inciter davantage de développeurs à apprendre et à utiliser TypeScript.

Exemples JavaScript <-> TypeScript

- Voici quelques exemples de code JavaScript et de leurs équivalents TypeScript :

- i. **Exemple 1 : Déclaration de variable**

- JavaScript :

```
let x = 5;
```

- TypeScript :

```
let x: number = 5;
```

- Dans TypeScript, vous pouvez spécifier le type de variable lors de sa déclaration.

i. Exemple 2 : les tableaux

■ JavaScript :

```
let desNombres = [1, 2, 3, 4, 5];
```

■ TypeScript :

```
let desNombres: number[] = [1, 2, 3, 4, 5];
```

i. Exemple 3 : Fonction

■ JavaScript :

```
function add(a, b) {  
    return a + b;  
}
```

■ TypeScript :

```
function add(a: number, b: number)  
: number {  
    return a + b;  
}
```

■ Dans TypeScript, vous pouvez spécifier les types des paramètres et le type de retour de la fonction.



i. Exemple 4 : Objet

■ JavaScript :

```
let user = {  
  name: "John",  
  age: 30  
};
```

■

■ TypeScript :

```
let user: {  
  name: string,  
  age: number  
}  
=  
{  
  name: "John",  
  age: 30  
};
```

i. Exemple 5 : Classes :

■ JavaScript :

```
class Car {  
  constructor(marque,  
modele) {  
    this.marque = marque;  
    this.modele = modele;  
  }  
  startEngine() {  
return 'Vroom!'; }  
}
```

■ TypeScript :

```
class Car {  
  marque: string;  
  modele: string;
```

```
  constructor(marque  
string, modele: string) {  
    this.marque = marque;  
    this.modele = modele;  
  }
```

```
  startEngine(): string {  
    return 'Vroom!';  
  }  
}
```

- Notez que TypeScript est un sur-ensemble strict de JavaScript,
 - ce qui signifie que chaque programme JavaScript valide
 - est également un programme TypeScript valide.

- Les ajouts de TypeScript, tels que les annotations de type et les interfaces, fournissent un typage statique optionnel pour JavaScript, ce qui peut aider à la détection des erreurs et à l'autocomplétion dans les éditeurs de code.

Configuration du projet

tsconfig

- Le fichier **tsconfig.json** est utilisé pour configurer le compilateur TypeScript pour un projet TypeScript.
- Il spécifie les options du compilateur à utiliser lors de la compilation du code TypeScript.

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "esnext",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "jsx": "react",
    "sourceMap": true,
    "esModuleInterop": true,
    "strict": true,
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "noImplicitAny": false,
    "noImplicitReturns": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true
  },
  "include": ["src"],
  "exclude": ["node_modules", "build", "dist", "public"]
}
```

b) la version de javascript

Cette ligne indique que notre code va être « compilé » en javascript ES6 :

```
3      "target": "es6",
```

c) option de « compilation »

```
20     "jsx": "react",
```

Dans un fichier **tsconfig.json** de TypeScript, l'option **"jsx": "react"** indique au compilateur TypeScript de transformer chaque balise JSX en un appel à la fonction **React.createElement**.

d) Les répertoires

```
"include": ["src"],  
"exclude": ["node_modules", "build", "dist", "public"]  
}
```

Node.JS

■ Node.js est une plateforme logicielle open-source qui exécute du JavaScript côté serveur. C'est une nécessité pour le développement avec React pour plusieurs raisons :

1. **Gestion des paquets** : Node.js vient avec npm (node package manager), qui est utilisé pour gérer les dépendances de votre projet.
2. **Outils de développement**
3. **Environnement de développement**
4. **Rendu côté serveur**

- TSX -> TypeScript
- **JSX** est une extension de syntaxe pour JavaScript dans le cadre de la bibliothèque React.
- Elle permet d'écrire ce qui ressemble à du HTML (ou XML) directement dans des fichiers JavaScript.
- **JSX** n'est pas compris par les navigateurs directement.
- Avant que le code puisse être exécuté dans le navigateur, il doit être transpilé (la transpilation) en JavaScript traditionnel.
- Cela est généralement accompli à l'aide de Babel, un outil de transpilation qui convertit le code JSX (et éventuellement moderne ES6+ JavaScript) en un JavaScript compatible avec les navigateurs actuels.

JSX (suite)

■ Exemple

```
const element = <h1>Hello, world!</h1>;
```

- Dans cet exemple, la balise `<h1>` est écrite directement dans le code JavaScript à l'aide de JSX. Babel convertira ce code en JavaScript pur pour créer un élément DOM qui peut être affiché dans le navigateur.

Les fichiers


APP.TSX

```
import React, { FunctionComponent } from 'react';
```

```
const App: FunctionComponent = () => {  
  const name: String = 'React';
```

```
  return (  
    <h1>Avec React : Hello {name} !</h1>  
  )  
}
```

```
export default App;
```



importe le module React
et le type
FunctionComponent
du module 'react'.

```
import React, { FunctionComponent } from 'react';
```

```
const App: FunctionComponent = () => {  
  const name: String = 'React';
```

```
  return (  
    <h1>Avec React : Hello {name} !</h1>  
  )  
}
```

```
export default App;
```

Le type **FunctionComponent** (ou **FC** en abrégé) est un type générique (c'est-à-dire un type qui peut prendre un autre type comme argument) utilisé pour typer les composants fonctionnels en React.

Définition de notre composant

```
import React, { FunctionComponent } from 'react';
```

```
const App: FunctionComponent = () => {  
  const name: String = 'React';
```

```
  return (  
    <h1>Avec React : Hello {name} !</h1>  
  )  
}
```

```
export default App;
```



Définition

Définition de notre composant

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

App est défini comme un composant fonctionnel qui retourne une balise **h1** avec le texte "Hello, React !". **name** est une constante de type **String** définie à l'intérieur de la fonction.

Définition de notre composant

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

Déclaration d'une constante

-> non modifiable

Définition de notre composant

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

App est utilisé pour stocker le composant fonctionnel React.

Définition de notre composant

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

la constante **App** est du type
FunctionComponent
-> un type spécifique
fourni représenter un
composant fonctionnel
React.

Définition de notre composant

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

C'est la définition de la fonction.

En JS et TS,

`() => {}` définit une fonction anonyme

Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

Rendu dans le DOM



Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

- utilisé pour spécifier la valeur de sortie d'une fonction.
- Dans un composant fonctionnel React,
- **ce que vous retournez est ce = ce qui sera rendu à l'écran**
- lorsque le composant est utilisé.

Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

Ceci est du JSX, qui est une extension de la syntaxe JavaScript qui ressemble à du HTML.

Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

Balise

Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: string = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```

Dans le JSX, les accolades `{}` sont utilisées pour intégrer des expressions JavaScript.

Ici, `{name}` insère la valeur de la variable **name** dans le JSX.

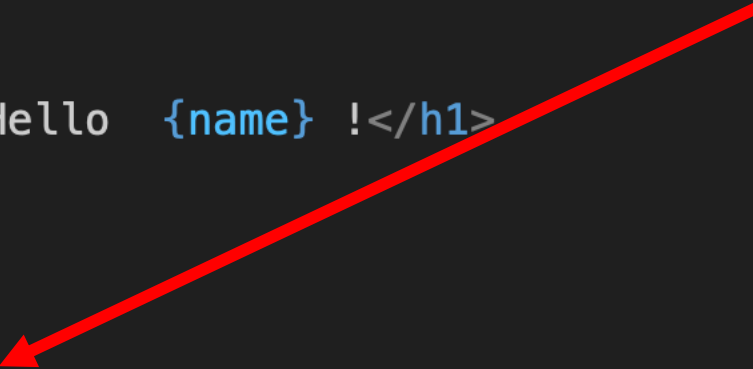
Rendu dans le DOM

```
import React, { FunctionComponent } from 'react';

const App: FunctionComponent = () => {
  const name: String = 'React';

  return (
    <h1>Avec React : Hello {name} !</h1>
  )
}

export default App;
```



exporte le composant **App** comme export par défaut du module.

Cela signifie qu'il peut être importé dans un autre fichier avec l'instruction **import App from './App';**

Les hooks

useState

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

, le Hook **useState** est utilisé pour ajouter un état local au composant fonctionnel **Compter**.

```
import React, { useState } from 'react';
```

```
function Compter() {  
  const [compteur, setCompteur]
```

L'état est une valeur qui est maintenue par le composant et qui peut changer au fil du temps.

Chaque fois que l'état change, le composant est rerendu avec la nouvelle valeur de l'état.

```
  return (  
    <div>
```

```
      <p>Vous avez cliqué {compteur} fois</p>
```

```
      <button onClick={() => setCompteur(compteur + 1)}>
```

```
        Cliquez moi
```

```
      </button>
```

```
    </div>
```

```
  );
```

```
}
```

```
export default Compter;
```

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

: **useState** est appelé avec la valeur initiale **0**. Cette fonction renvoie un tableau de deux éléments.

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

Le premier élément est la valeur actuelle de l'état (**compteur**),

et le deuxième élément est une fonction (**setCompteur**) qui permet de mettre à jour cet état.

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

Dans ce cas, **compteur** est la valeur actuelle de l'état

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}


export default Compter;
```

et **setCompteur** est la fonction qui permet de mettre à jour cet état

```
import React, { useState } from 'react';

function Compter() {
  const [compteur, setCompteur] = useState(0);

  return (
```



C'est donc React qui fournit et définit **setCompteur** lorsque vous appelez **useState**.

La fonction **setCompteur** peut être appelée avec une nouvelle valeur pour mettre à jour l'état **compteur**.

Lorsqu'elle est appelée, React

- mettra à jour l'état **compteur** avec la nouvelle valeur
- et réexécutera le composant avec la nouvelle valeur de l'état.

```
export default Compter;
```



```
import React, { useState } from 'react'

function Compter() {
  const [compteur, setCompteur] = useState(0)

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

appelle la fonction **setCompteur** avec la valeur actuelle de **compteur** plus 1.

Cela mettra à jour l'état **compteur** et **forcera** le composant à être rerendu avec la nouvelle valeur de **compteur**.

```
import React, { useState } from 'react'

function Compter() {
  const [compteur, setCompteur] = useState(0)

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

appelle la fonction **setCompteur** avec la valeur actuelle de **compteur** plus 1.

Cela mettra à jour l'état **compteur** et **forcera** le composant à être rerendu avec la nouvelle valeur de **compteur**.

```
import React, { useState } from 'react'

function Compter() {
  const [compteur, setCompteur] = useState(0)

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Compter;
```

appelle la fonction **setCompteur** avec la valeur actuelle de **compteur** plus 1.

Cela mettra à jour l'état **compteur** et **forcera** le composant à être rerendu avec la nouvelle valeur de **compteur**.

useEffect

```
import React, { useState, useEffect } from 'react';

function Exemple() {
  const [compteur, setCompteur] = useState(0);

  // Similaire à componentDidMount et componentDidUpdate
  useEffect(() {
    // Mettre à jour le titre du document en utilisant l'API d
    document.title = `Vous avez cliqué ${compteur} fois`;
  }, [compteur]); // N'exécute l'effet que si compteur change

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Exemple;
```

useEffect est utilisé pour effectuer un effet de bord après chaque rendu.

```
import React, { useState, useEffect } from 'react';

function Exemple() {
  const [compteur, setCompteur] = useState(0);

  // Similaire à componentDidMount et componentDidUpdate :
  useEffect(() => {
    // Mettre à jour le titre du document en utilisant l'API d
    document.title = `Vous avez cliqué ${compteur} fois`;
  }, [compteur]); // N'exécute l'effet que si compteur change

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Exemple;
```

Le premier argument de **useEffect** est une fonction qui contient le code de l'effet à exécuter.

```
import React, { useState, useEffect } from 'react';

function Exemple() {
  const [compteur, setCompteur] = useState(0);

  // Similaire à componentDidMount et componentDidUpdate :
  useEffect(() => {
    // Mettre à jour le titre du document en utilisant l'API d
    document.title = `Vous avez cliqué ${compteur} fois`;
  }, [compteur]); // N'exécute l'effet que si compteur change

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Exemple;
```

Le deuxième argument de **useEffect** est un tableau de dépendances.

C'est une liste de valeurs que l'effet utilise et pour lesquelles il doit être réexécuté lorsqu'elles changent. .

```
import React, { useState, useEffect } from 'react';

function Exemple() {
  const [compteur, setCompteur] = useState(0);

  // Similaire à componentDidMount et componentDidUpdate :
  useEffect() => {
    // Mettre à jour le titre du document en utilisant l'API d
    document.title = `Vous avez cliqué ${compteur} fois`;
  }, [compteur]); // N'exécute l'effet que si compteur change

  return (
    <div>
      <p>Vous avez cliqué {compteur} fois</p>
      <button onClick={() => setCompteur(compteur + 1)}>
        Cliquez moi
      </button>
    </div>
  );
}

export default Exemple;
```

Par défaut, le Hook **useEffect** est appelé après le premier rendu du composant, c'est-à-dire au démarrage de l'application

ou plus précisément lorsque le composant est monté pour la première fois

Plus élaboré

```

import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  },
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>Nombre de super-héros chargés: {heroes.length}</p>
      {heroes.map((hero: SuperHero) => (
        <div key={hero.id}>
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>
          {
            <img src={`https://cdn.jsdelivr.net/gh/rtomczak/superhero-
api@0.3.0/api/images/sm/${hero.slug}.jpg`} alt={hero.name} />
          }
        </div>
      ))}
    </div>
  );
}

export default App;

```

Ce script React TypeScript définit le composant **App**, qui utilise un tableau de données **SuperHero** pour afficher une liste de super-héros.

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHeroesData from './SuperHeroes.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHeroesData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>Nombre de super-héros chargés: {heroes.length}</p>
      {heroes.map((hero: SuperHero) => (
        <div key={hero.id}>
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>
        </div>
      ))}
    </div>
  );
}
```

Chaque super-héros est
défini par une classe
SuperHero

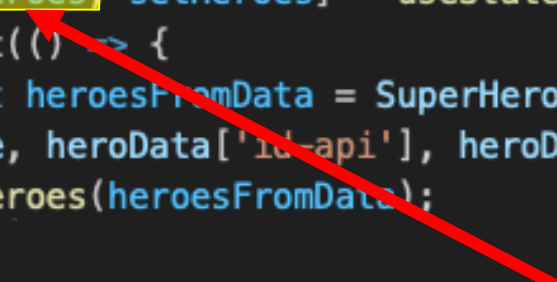
```
import React, { useState, useEffect } from 'react';  
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero  
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON
```

```
export const App = () => {  
  const [heroes, setHeroes] = useState<SuperHero[]>([]);  
  useEffect(() => {  
    const heroesFromData = SuperHerosData.map((heroData: any)  
heroData.name, heroData['id-api'], heroData.slug));  
    setHeroes(heroesFromData);  
  }, []);  
  return (  
    <div>  
      <h1>Super Heroes App</h1>  
      <p>Nombre de super-héros chargés: {heroes.length}</p>  
      {heroes.map((hero: SuperHero) => (  
        <div key={hero.id}>  
          <h2>{hero.name}</h2>  
          <p>Id: {hero.id}</p>  
          <p>Id API: {hero.idApi}</p>  
          <p>Slug: {hero.slug}</p>
```

les données de ces
super-héros sont
chargées à partir d'un
fichier JSON.

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
```



Ici, un nouvel état local **heroes** est déclaré à l'aide du Hook **useState**.

```
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div
```

Ici, un nouvel état local **heroes** est déclaré à l'aide du Hook **useState**.

```
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div
```

Ici, un nouvel état local **heroes** est déclaré à l'aide du Hook **useState**.

Cet état représente un tableau d'objets **SuperHero**.

```
    <div key={hero.id}>
      <h2>{hero.name}</h2>
      <p>Id: {hero.id}</p>
      <p>Id API: {hero.idApi}</p>
      <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map(heroData => new SuperHero(heroData.i
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div
```

Ici, un nouvel état local **heroes** est déclaré à l'aide du Hook **useState**.

Cet état représente un tableau d'objets **SuperHero**.

Initialement, cet état est défini comme un tableau vide.

```
<p>Slug: {hero.slug}</p>
```



```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHeroesData from './SuperHeroes.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHeroesData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div
```

Ici, un nouvel état local **heroes** est déclaré à l'aide du Hook **useState**.

Cet état représente un tableau d'objets **SuperHero**.

Initialement, cet état est défini comme un tableau vide.

setHeroes est la fonction qui sera utilisée pour mettre à jour cet état

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
```

Le Hook **useEffect** est utilisé pour effectuer une action après le rendu du composant.

```
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.id,
    heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div
```

Dans notre cas, il est utilisé pour effectuer une action lorsque le composant est **monté pour la première fois**.

```
{heroes.map((hero: SuperHero) => (
  <div key={hero.id}>
    <h2>{hero.name}</h2>
    <p>Id: {hero.id}</p>
    <p>Id API: {hero.idApi}</p>
    <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);

  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  },
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>

```

. Ici, il est utilisé pour initialiser l'état **heroes** avec les données du fichier JSON.

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHeroesData from './SuperHeroes.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHeroesData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h

```

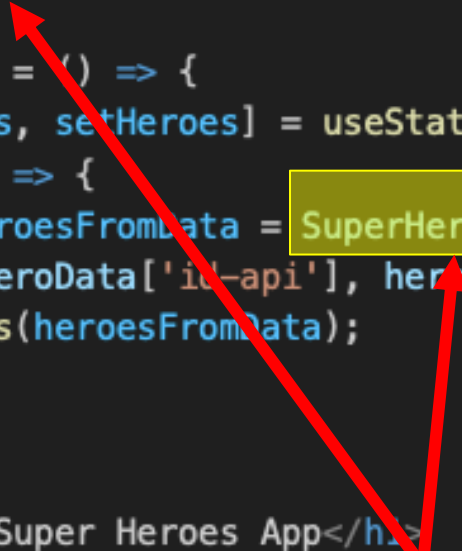
. La fonction **map** est utilisée pour convertir chaque objet du fichier JSON en une instance de **SuperHero**.

```
<p>Id: {hero.id}</p>
<p>Id API: {hero.idApi}</p>
<p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHeroesData from './SuperHeroes.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHeroesData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h

```



A red arrow originates from the `SuperHeroesData` variable in the `import` statement, points to the `SuperHeroesData.map` call inside the `useEffect` function, and then points to the `{h` in the `<p>` JSX element.

. La fonction **map** est utilisée pour convertir chaque objet du fichier JSON en une instance de **SuperHero**.

```
<p>Id: {hero.id}</p>
<p>Id API: {hero.idApi}</p>
<p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON


export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
```

Cette parcourt **chaque élément** du tableau et **applique** une fonction à chaque élément, puis **retourne** un nouveau tableau avec les résultats.

```
      <p>Id: {hero.id}</p>
      <p>Id API: {hero.idApi}</p>
      <p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {heroes.map(hero => SuperHero, {})}
        <div key={hero.id}>
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>
        </div>
      </p>
    </div>
  );
}
```

A red arrow originates from a white text box with a black border and points to the argument `(heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug)` passed to the `map` function in the `useEffect` hook. The text box contains the French text: "La fonction passée à **map** est une fonction fléchée".

La fonction passée à **map** est une fonction fléchée


```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHeroesData from './SuperHeroes.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHeroesData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>

```

La fonction passée à **map** est une fonction fléchée qui prend un élément **heroData** du tableau .

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>

```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
```

Ainsi la fonction passée à **map** est une fonction fléchée qui prend un élément **heroData** du tableau **SuperHerosData** comme argument et retourne une nouvelle instance de la classe **SuperHero**,

```
<p>Id API: {hero.idApi}</p>
<p>Slug: {hero.slug}</p>
```

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>
        {h
          <h2>{hero.name}</h2>
          <p>Id: {hero.id}</p>
          <p>Id API: {hero.idApi}</p>
          <p>Slug: {hero.slug}</p>

```

Ensuite, **setHeroes** est appelé avec le tableau nouvellement créé pour mettre à jour l'état **heroes**.

```
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON

export const App = () => {
  const [heroes, setHeroes] = useState<SuperHero[]>([]);
  useEffect(() => {
    const heroesFromData = SuperHerosData.map((heroData: any) => new SuperHero(heroData.name, heroData['id-api'], heroData.slug));
    setHeroes(heroesFromData);
  }, []);
  return (
    <div>
      <h1>Super Heroes App</h1>
      <p>Nombre de super-héros chargés: {heroes.length}</p>
      {heroes.map((hero: SuperHero) => (
        <div key={hero.id}>
          <h2>{hero.name}</h2>
          <p>{hero.id}</p>
          <p>{hero.slug}</p>
        </div>
      ))}
    </div>
  )
}
```

Le tableau vide en deuxième argument signifie que cet effet ne s'exécute qu'une fois, lors du montage du composant.

```

return (
  <div>
    <h1>Super Heroes App</h1>
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
        {
          <img src={`https://cdn.jsdelivr.net/gh/rtomczak/superhero-
api@0.3.0/api/images/sm/
        }
      </div>
    ))}
  </div>
);
}
export default App;

```

Dans un composant fonctionnel React, ce que vous retournez est ce qui sera rendu à l'écran lorsque le composant est utilisé.

```

return (
  <div>
    <h1>Super Heroes App</h1>
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
        {
          <img src={`https://cdn.jsdelivr.net/gh/rtomczak/superhero-
api@0.3.0/api/images/sm/
        }
      </div>
    ))}
  </div>
);
}
export default App;

```

La taille du tableau

```

return (
  <div>
    <h1>Super Heroes App</h1>
    <p>Nombre de super-héros chargés: {heroes.length}</p>
    {heroes.map((hero: SuperHero) => (
      <div key={hero.id}>
        <h2>{hero.name}</h2>
        <p>Id: {hero.id}</p>
        <p>Id API: {hero.idApi}</p>
        <p>Slug: {hero.slug}</p>
        {
          <img src={`https://cdn.jsdelivr.net/gh/rtomczak/superhero-
api@0.3.0/api/images/sm/
        }
      </div>
    ))}
  </div>
);
};
}
export default App;

```

Encore une fonction fléchée sur map

Qui va pour chaque élément

```
return (  
  <div>  
    <h1>Super Heroes App</h1>  
    <p>Nombre de super-héros chargés: {heroes.length}</p>  
    {heroes.map((hero: SuperHero) => (  
      <div key={hero.id}>  
        <h2>{hero.name}</h2>  
        <p>Id: {hero.id}</p>  
        <p>Id API: {hero.idApi}</p>  
        <p>Slug: {hero.slug}</p>  
        {  
          <img src={`https://cdn.jsdelivr.net/gh/rtomczak/superhero-  
api@0.3.0/api/images/sm/${hero.slug}.jpg`} alt={hero.name} />  
        }  
      </div>  
    )  
  )}  
  </div>  
);  
}  
export default App;
```

Clic.tsx

4. *Clic.tsx*

```
import React, { FunctionComponent, useState } from 'react';

const App: FunctionComponent = () => {
  const [name, setName] = useState('World');
  return (
    <div>
      <h1>Avec React : Hello {name}!</h1>
      <button onClick={() => setName(name + "!")}>
        Ajoutez un !
      </button>
      <button onClick={() => setName("World")}>
        Remettre World
      </button>
    </div>
  )
}

export default App;
```

superHeros.ts

■ Les fichiers **.ts** et **.tsx** sont tous deux des fichiers TypeScript, mais il y a une différence clé entre eux :

- Les fichiers **.ts** sont des fichiers TypeScript standard. Ils peuvent contenir n'importe quel code TypeScript valide.
- Les fichiers **.tsx** sont des fichiers TypeScript qui peuvent également contenir du JSX. JSX est une extension de syntaxe pour JavaScript qui ressemble à XML ou HTML. Il est couramment utilisé avec des bibliothèques comme React pour décrire la structure de l'interface utilisateur.

```
export class SuperHero {  
  id: number;  
  name: string;  
  idApi: number;  
  slug: string | undefined;  
  
  constructor(id: number, name: string, idApi: number, slug?: string) {  
    this.id = id;  
    this.name = name;  
    this.idApi = idApi;  
    this.slug = slug;  
  }  
}
```

Dans cette classe **SuperHero**, il y a quatre propriétés : **id**, **name**, **idApi**, et **slug**.

```
export class SuperHero {  
  id: number;  
  name: string;  
  idApi: number;  
  slug: string | undefined;  
  
  constructor(id: number, name: string, idApi: number, slug?: string) {  
    this.id = id;  
    this.name = name;  
    this.idApi = idApi;  
    this.slug = slug;  
  }  
}
```

, le constructeur est une méthode spéciale dans la classe, utilisée pour créer et initialiser un nouvel objet.

```
export class SuperHero {  
  id: number;  
  name: string;  
  idApi: number;  
  slug: string | undefined;  
  
  constructor(id: number, name: string, idApi: number, slug?: string) {  
    this.id = id;  
    this.name = name;  
    this.idApi = idApi;  
    this.slug = slug;  
  }  
}
```

Quand un nouvel objet **SuperHero** est créé, le constructeur est appelé avec les valeurs **id**, **name**, **idApi**, et **slug** fournies, qui sont ensuite assignées aux propriétés correspondantes de l'objet.


```
export class SuperHero {
```

```
  id: number;
```

```
  name: string;
```

```
  idApi: number;
```

```
  slug: string | undefined;
```

```
  constructor(id: number, name: string, idApi: number, slug?: string) {
```

```
    this.
```

```
    this.r
```

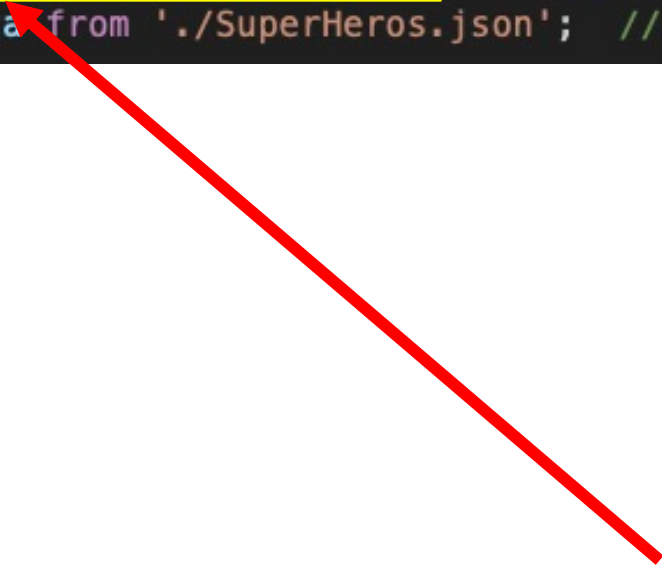
```
    this.i
```

```
    this.s
```

```
  }
```


Enfin, cette classe est exportée avec le mot-clé **export**, ce qui signifie qu'elle peut être importée et utilisée dans d'autres fichiers TypeScript dans le même projet.

```
// App.tsx
import React, { useState, useEffect } from 'react';
import { SuperHero } from './SuperHero'; // Importation de notre classe SuperHero
import SuperHerosData from './SuperHeros.json'; // Importation du fichier JSON
```



Index.tsx


Tout d'abord, il importe les bibliothèques nécessaires. **React** est importé de "react", qui est la bibliothèque principale de React



```
import React from "react";
import ReactDOM from "react-dom";
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

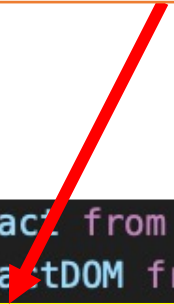
ReactDOM est importé de "react-dom", qui est une bibliothèque permettant de manipuler le DOM (Document Object Model) avec React



```
import React from "react";
import ReactDOM from "react-dom";
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

Enfin, le composant **App** est importé du fichier **App**.



```
import React from "react";
import ReactDOM from "react-dom";
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

```
import React from "react";
import ReactDOM from "react-dom";
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

Ensuite, la méthode **ReactDOM.render** est utilisée pour rendre le composant **App** dans l'élément DOM

```
import React from "react";  
import ReactDOM from "react-dom";  
import App from './App'
```

```
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
)
```

Ensuite, la méthode **ReactDOM.render** est utilisée pour rendre le composant **App** dans l'élément DOM avec l'id 'root'.

. L'appel à **ReactDOM.render** prend deux arguments : le premier est ce que vous voulez rendre (dans ce cas, le composant **App**),

```
import  
import ReactDOM from "react-dom";  
import App from './App'  
  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
)
```


```
import React from "react";
import ReactDOM from "react-dom";
import App from './App'

ReactDOM.render(
  <App />,
  document.getElementById('root')
)
```

et le deuxième est où vous voulez le rendre (dans ce cas, l'élément DOM avec l'id 'root').

```
import React from "react";  
import ReactDOM from "react-dom";  
import App from './App'
```

```
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
)
```



signifie donc "Rendez le composant **App** à l'intérieur de l'élément DOM ayant l'id 'root'". En pratique, cela signifie que votre application React sera affichée à l'intérieur de cet élément 'root'.

FIN