

# React

## Installation

[NodeJS](#) requis

### Standalone project

Création du projet :

```
npx create-react-app my-app
```

Eventuellement avec le support TypeScript :

```
npx create-react-app my-app --template typescript
```

Démarrage du serveur

```
cd my-app  
npx start
```

## Composants



En React, tout est composant, et les composants peuvent être créés à partir de fonctions (JS), ou de classes (ES6 ou TypeScript).

## Fonctions

Un composant peut être créé à partir d'une fonction :

```
function Hello() {  
  return (  
    <div>  
      <h1>props.message</h1>  
    </div>  
  );  
}  
  
export default Hello;
```

## Classe

Un composant peut être créé à partir d'une classe ES6 :

```
import React from "react";

class Hello extends React.Component {
  render() {
    return <h1>{this.props.message}</h1>;
  }
}

export default Hello;
```

## Classe TypeScript

Un composant peut être créé à partir d'une classe TypeScript:

```
import React from "react";

export default class Hello extends React.Component<{message: string}> {
  render() {
    return (
      <h1>{this.props.message}</h1>
    );
  }
}
```

La classe Hello déclare explicitement une propriété de nom **message** de type **string**.

## Utilisation

Utilisation de **Hello** dans l'**App**, et initialisation en JSX de la prop **message** avec l'attribut **message** :

```
import Hello from './hello';

function App() {
  return (
    <div className="App">
      <Hello message="Hello World!" />
    </div>
  );
}

export default App;
```

## Props et state

Sur un composant React :

- Les propriétés sont accessibles avec l'objet **props** passé en paramètre (du constructeur ou de la fonction)
- Ces propriétés sont initialisées en JSX/HTML via les attributs de l'élément.
- Elles sont immutables (en lecture seule).
- Il est nécessaire d'utiliser l'objet **state** pour modifier l'état d'un composant.

### Exemple classe TypeScript

```
import React from "react";

export default class Hello extends React.Component<{message:string},{msg: string}>
{
  constructor(props: {message:string}) {
    super(props);
    this.state = {msg: props.message};
  }
  handleChange = (event: React.ChangeEvent<HTMLInputElement>) => {
    this.setState({msg: event.target.value})
  }
  render() {
    return (
      <>
        <h1>{this.state.msg}</h1>
        <input value={this.state.msg} onChange={this.handleChange}/>
      </>
    );
  }
}
```

### Exemple fonction JavaScript

```
import {useState} from "react";

export default function Hola(props){
  const [message, setMessage] = useState(props.message);
  const handleChange = (e) => {
    setMessage(e.target.value);
  }
  return (
    <>
      <h1>{message}</h1>
      <input type="text" value={message} onChange={handleChange}/>
    </>
  );
}
```

## Http Wrapper

```
// This is a wrapper for the fetch API
export default class AppHttp {
  static async get(url, options) {
    const response = await fetch(url, options);
    return await response.json();
  }
  static async post(url, body, options) {
    const response = await fetch(url, {
      method: 'POST',
      body: JSON.stringify(body),
      ...options
    });
    return await response.json();
  }
  static async put(url, body, options) {
    const response = await fetch(url, {
      method: 'PUT',
      body: JSON.stringify(body),
      ...options
    });
    return await response.json();
  }
  static async delete(url, options) {
    const response = await fetch(url, {
      method: 'DELETE',
      ...options
    });
    return await response.json();
  }
}
```

## Contexte

```
'use client';
import React, {createContext, ReactNode, useContext, useState} from 'react';

interface DemoContextType {
  data: number;
  setData: (d:number)=>void;
}

const DemoContext = createContext<DemoContextType | undefined>(undefined);

export const demoProvider: React.FC<{ children: ReactNode }> = ({children}) => {
  const [data, setData] = useState<number>(0);
```

```
    return (
      <DemoContext.Provider value={{data, setData}}>
        {children}
      </DemoContext.Provider>
    );
  };

export const useDemo = (): DemoContextType => {
  const context = useContext(demoContext);
  if (!context) {
    throw new Error('usedemo must be used within a DemoProvider');
  }
  return context;
};
```

## Hooks React

Les hooks React sont des fonctions introduites à partir de React 16.8 permettant d'utiliser l'état et d'autres fonctionnalités de React dans les composants fonctionnels (basés sur une fonction et non une classe).

### Hooks de base

## useState

Permet de gérer un état local dans un composant fonctionnel.

```
import { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0); // Déclare un état `count`

  return (
    <button onClick={() => setCount(count + 1)}>Compteur : {count}</button>
  );
}
```

## useEffect

Permet d'exécuter des effets secondaires dans un composant. Exemples :

- appels API,
- écouteurs d'événements,
- manipulation du DOM...

```
import { useState, useEffect } from "react";

function Timer() {
```

```
const [seconds, setSeconds] = useState(0);

useEffect(() => {
  const interval = setInterval(() => {
    setSeconds(s => s + 1);
  }, 1000);

  return () => clearInterval(interval); // Nettoyage de l'effet
}, []); // Dépendances vides = effet exécuté une seule fois au montage

return <p>Temps écoulé : {seconds} sec</p>;
}
```

## useContext

Permet d'utiliser un contexte global sans avoir à passer des props manuellement à chaque niveau.

```
import { useContext, createContext } from "react";

const ThemeContext = createContext("light");

function ThemedButton() {
  const theme = useContext(ThemeContext); // Récupère la valeur du contexte

  return <button style={{ background: theme === "dark" ? "#333" : "#ddd" }}>
    Thème : {theme}
  </button>;
}
```

## Composants

- [React-hook-form](#)
- [React-query](#)

From:

<http://slamwiki2.kobject.net/> - **SlamWiki 2.1**

Permanent link:

<http://slamwiki2.kobject.net/web/framework/react?rev=1739923861>

Last update: **2025/08/12 02:35**

