



9. Personnaliser les composants React

9.1. Styliser les composants React


Il existe différentes manières de styler les composants React.

Inline CSS

Il s'agit du style CSS envoyé à l'élément directement via HTML ou JSX. Vous pouvez inclure un objet JavaScript pour CSS dans les composants React.

Exemple

```
function Button() {
  return (
    <button style={{color:"#fff", borderColor:"#5C5CFF",
padding:"10px", backgroundColor:"#5C5CFF" }}>Nom du bouton</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Button />
);
```



Modules CSS

Les modules CSS s'assurent que tous les styles d'un composant sont regroupés à un seul endroit et s'appliquent à ce composant particulier. Cela résout certainement le problème de portée globale du CSS. La fonctionnalité de composition agit comme une arme pour représenter les styles partagés entre les états.

Exemple


Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
function Button() {
  return (
    <button className='buttonStyle'>Nom du bouton</button>
  );
}
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Button />
);
```



Index.css

```
.buttonStyle {  
  color: #fff;  
  border-color: #5C5CFF;  
  padding:10px;  
  background-color:#5C5CFF;  
}
```



Composants stylisés

Les composants stylisés sont l'une des nouvelles façons d'utiliser CSS dans le JavaScript moderne. C'est censé être le successeur des modules CSS, un moyen d'écrire du CSS qui est limité à un seul composant, et ne fuit pas vers un autre élément de la page

Les composants stylisés vous permettent d'écrire du CSS brut dans vos composants sans vous soucier des collisions de noms de classe.

Installer des composants stylisés en utilisant npm :

```
npm install styled-components
```

C'est tout ! Il ne vous reste plus qu'à ajouter cette importation :

```
import styled from 'styled-components'
```


Avec le styled objet importé, vous pouvez maintenant commencer à créer des composants stylisés. Voici le premier:

```
const Button = styled.button`  
  color: #fff;  
  border-color: #5C5CFF;  
  padding:10px;  
  background-color:#5C5CFF;  
`
```

Maintenant, ce composant peut être rendu dans notre conteneur en utilisant la syntaxe normale de React:

```
return <Button >Nom du bouton</Button>;
```

Les composants stylisés offrent d'autres fonctions que vous pouvez utiliser pour créer d'autres composants, non seulement button, aimer section, h1, input et plein d'autres.



Il existe d'autres façons pour styler les éléments tels que CSS dans JS, sass & SCSS et Less.



9.2. Affichage conditionnel

L'affichage conditionnel en React fonctionne de la même façon que les conditions en Javascript. On utilise l'instruction Javascript **if** ou l'**opérateur ternaire** pour créer des éléments représentant l'état courant, et on laisse React mettre à jour l'interface utilisateur (UI) pour qu'elle corresponde.

Considérons ces deux composants :


Connecte.js	Anonymous.js
<pre>function Connecte() { return (<h1>Bienvenue !</h1>); } export default Connecte;</pre>	<pre>function Anonymous() { return (<h1>Veuillez vous inscrire.</h1>); } export default Anonymous;</pre>

Nous allons créer un composant App qui affiche un de ces deux composants, selon qu'un utilisateur est connecté ou non :

```
import Anonymous from './Anonymous';
import Connecte from './Connecte';

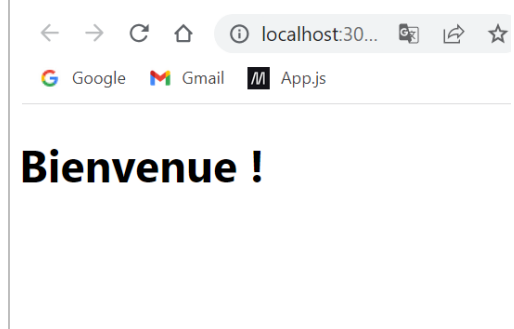
function App(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <Connecte />;
  }
  return <Anonymous />;
}
export default App;
```

Cet exemple affiche un message différent selon la valeur de la prop isLoggedIn.

Code	Affichage
<pre>const root = ReactDOM.createRoot(document.getElemen tById('root')); root.render(<React.StrictMode> <App isLoggedIn={false} /> </React.StrictMode>);</pre>	 <p>The screenshot shows a browser window at localhost:30... with search engines for Google, Gmail, and App.js. The main content area displays the text "Veuillez vous inscrire." in a large, bold, black font.</p>



```
const root =
ReactDOM.createRoot(document.getElementBy
tById('root'));
root.render(
  <React.StrictMode>
    <App isLoggedIn={true} />
  </React.StrictMode>
);
```



9.3. Listes et clés

Les listes

La méthode **map()** est utilisé pour prendre un tableau de nombres et doubler leurs valeurs. On peut construire des collections d'éléments et les inclure dans du JSX en utilisant les accolades {}.

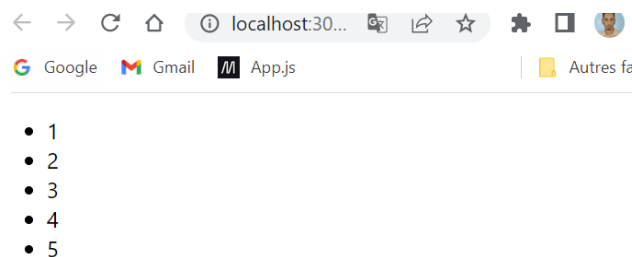
Ci-dessous, on itère sur le tableau de nombres en utilisant la méthode JavaScript map(). On retourne un élément pour chaque entrée du tableau. Enfin, on affecte le tableau d'éléments résultant à listItems :

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((numbers) =>
  <li>{numbers}</li>
);
```

On inclut tout le tableau listItems dans un élément , et on l'affiche dans le DOM :

```
const root = ReactDOM.createRoot(document.getElementBy
Id('root'));
root.render(
  <ul>{listItems}</ul>
);
```

Ce code affiche une liste à puces de nombres entre 1 et 5.





Les clés

Les clés aident React à identifier quels éléments d'une liste ont changé, ont été ajoutés ou supprimés. Vous devez donner une clé à chaque élément dans un tableau afin d'apporter aux éléments une identité stable :

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

Le meilleur moyen de choisir une clé est d'utiliser quelque chose qui identifie de façon unique un élément d'une liste parmi ses voisins. Le plus souvent on utilise l'ID de notre donnée comme clé :

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```