





Composants et props

Thèmes	 React
Description	Introduction au système de props dans React.
Date de création	@13 mars 2025 23:08
Créée par	 Kévin Wolff

Chapitres

 [Introduction](#)

 [Exemples](#)


 [Props variable](#)


 [Props fonction](#)

 [Props composant](#)

Sur le même sujet

 [Premier composant React](#)

 [Le router](#)

 [Approche atomic design](#)

Introduction

 [Documentation officielle - props system](#)

Les composants React utilisent les "props" pour communiquer entre eux. Un composant parent peut transmettre des informations (variables, objets, composants, fonctions, etc.) à ses composants enfants via les props. La syntaxe des props est similaire à celle des attributs HTML classiques.

Reprenons l'exemple du composant `Avatar` présenté dans  [Premier composant React](#).

```
export default function Avatar() {  
  return (  
      
  );  
}
```

Dans sa version initiale, ce composant n'est pas réutilisable, car les données de l'image sont déclarées en dur, ce qui rend l'image statique. En utilisant JSX et l'interpolation des variables, il est possible d'extraire ces données dans des variables pour les exploiter dans le balisage.

```
export default function Avatar() {  
  const avatarSrc = "https://aws.image.avatar.waglecorp.png";  
  const avatarAlt = "Avatar de Wagle Corp";
```

```

return (
  <img
    className="avatar"
    src={avatarSrc}
    alt={avatarAlt}
  />
);
}

```

Cependant, même en procédant ainsi, le composant reste figé. Pour le rendre dynamique et réutilisable, on utilise le système de props. Cela permet au composant de recevoir ses données (la source de l'image et son texte alternatif) depuis son composant parent.

Un composant React n'est rien d'autre qu'une fonction JavaScript. Comme toute fonction, il peut recevoir des paramètres. Voici les deux manières de déclarer ces paramètres.

```

export default function Avatar(props) {
  return (
    ...
  );
}

```

Cette syntaxe regroupe tous les paramètres reçus dans un objet `props` duquel vous pouvez exploiter les données de la manière suivante :

```

export default function Avatar(props) {
  return (
    <img
      className="avatar"
      src={props.avatarSrc}
      alt={props.avatarAlt}
    />
  );
}

```

```

export default function Avatar({ avatarSrc, avatarAlt }) {
  return (
    ...
  );
}

```

Dans le premier exemple tous les paramètres se trouvent dans un objet, avec cette syntaxe vous décomposez aussi tôt l'objet afin d'alléger la syntaxe.

```

export default function Avatar({ avatarSrc, avatarAlt }) {
  return (
    <img

```

```

    className="avatar"
    src={avatarSrc}
    alt={avatarAlt}
  />
);
}

```

Vous êtes libre d'adopter la syntaxe qui vous convient le mieux, en fonction du contexte et des habitudes de votre projet. Il n'existe pas de meilleure syntaxe, le choix repose principalement sur la lisibilité et la préférence personnelle.

Le composant `Avatar` est maintenant prêt à recevoir ses données (la source de l'image et son texte alternatif) depuis son composant parent. Il faut donc modifier le composant parent `UserProfileCard` pour lui transmettre ces données conformément aux attentes du composant enfant.

La syntaxe utilisée pour passer des props est similaire à celle des attributs HTML. Si vous avez opté pour la déstructuration des props dans le composant enfant, les noms des props doivent correspondre exactement aux noms des variables attendues.

```

import Avatar from './Avatar.jsx';
import UserInformation from './UserInformation.jsx';
import UserSocialMediaLinks from './UserSocialMediaLinks.jsx';

export default function UserProfileCard() {
  const avatarSrc = "https://aws.image.avatar.waglecorp.png";
  const avatarAlt = "Avatar de Wagle Corp";

  return (
    <div>
      <Avatar avatarSrc={avatarSrc} avatarAlt={avatarAlt} />
      <UserInformation />
      <UserSocialMediaLinks />
    </div>
  );
}

```

Félicitations ! Vous venez de rendre le composant `Avatar` réutilisable et autonome. Il reçoit désormais ses paramètres dynamiquement et les exploite pour produire un résultat. Cette approche suit une bonne pratique issue de la programmation fonctionnelle : concevoir des unités logiques modulaires et indépendantes qui prennent des paramètres en entrée et retournent un résultat. Cette règle s'applique autant aux fonctions qu'aux composants React.

Exemples

Les props permettent de transmettre n'importe quel type de données d'un composant parent à un composant enfant, en respectant la hiérarchie des composants. Voici des exemples d'usage des props avec différents types de données.

Props variable

```
export default function Avatar({ avatarSrc, avatarAlt }) {
  return (
    <img
      className="avatar"
      src={avatarSrc}
      alt={avatarAlt}
    />
  );
}
```

```
import Avatar from './Avatar.jsx';

export default function UserProfileCard() {
  const avatarSrc = "https://aws.image.avatar.waglecorp.png";
  const avatarAlt = "Avatar de Wagle Corp";

  return (
    <div>
      <Avatar avatarSrc={avatarSrc} avatarAlt={avatarAlt} />
    </div>
  );
}
```

Props fonction

```
export default function Avatar({ onClickFunction }) {
  return (
    
  );
}
```

```
import Avatar from './Avatar.jsx';

export default function UserProfileCard() {
  const sayHello = () => {
    console.log("Hello");
  }

  return (
    <div>
      <Avatar onClickFunction={sayHello} />
    </div>
  );
}
```

```
};  
}
```

Dans JSX, une fonction ne doit pas être appelée directement mais passée comme référence.
Voici un tableau récapitulatif pour bien comprendre la différence :

✅ Passer une fonction	❌ Appeler une fonction
<code><Avatar onClickFunction={sayHello} /></code>	<code><Avatar onClickFunction={sayHello()} /></code>
<code><button onClick={handleClick}></code>	<code><button onClick={handleClick()}></code>

Si vous écrivez `onClickFunction={sayHello()}`, la fonction sera immédiatement exécutée dès le rendu du composant, au lieu d'attendre l'événement `onClick`.

Si la fonction doit recevoir des arguments lors de son exécution, utilisez une fonction fléchée anonyme pour envelopper l'appel à la fonction.

```
export default function Avatar({ onClickFunction }) {  
  return (  
     message("Hello world!") }  
    />  
  );  
}
```

```
import Avatar from './Avatar.jsx';  
  
export default function UserProfileCard() {  
  const sayHello = (message) => {  
    console.log(message);  
  }  
  
  return (  
    <div>  
      <Avatar onClickFunction={sayHello} />  
    </div>  
  );  
}
```

Dans cet exemple, la fonction `sayHello` attend une propriété `message` qui sera ensuite affiché dans la console du devtools.

Pour pouvoir fournir ce message lors de l'exécution de la fonction, c'est à dire après que l'utilisateur ai cliqué sur son image de profil, l'appel à la fonction est "enveloppé" dans une fonction fléchée anonyme qui n'est donc pas appelée mais bien passée comme indiqué précédemment.

Props composant

Vous pouvez véhiculer un composant, d'un composant à un autre via les props de deux manières.

- Props classique : très rigide mais utile, pour passer une icône à un bouton par exemple.
- Children : même chose mais beaucoup plus flexible, utile pour la mise en page par exemple.

Props classique

Un composant d'image simple.

```
export default function Avatar() {  
  return (  
    <img ... />  
  );  
}
```

Un composant `SettingsPage` importe le composant d'image simple et le véhicule comme props au composant enfant `UserProfileCard`.

```
import Avatar from './Avatar.jsx';  
  
export default function SettingsPage() {  
  return (  
    <div>  
      <UserProfileCard AvatarComponent={<Avatar />} />  
    </div>  
  );  
}
```

```
export default function UserProfileCard({ AvatarComponent }) {  
  return (  
    <div>  
      < AvatarComponent />  
    </div>  
  );  
}
```

Le composant enfant `UserProfileCard` reçoit le composant qu'il doit afficher via les props fournis par le composant parent `SettingsPage`.

Children

Une autre manière de passer un composant enfant est d'utiliser la props spéciale `children`, ce qui offre plus de flexibilité et permet d'encapsuler le contenu dynamiquement.

Un composant `SettingsPage` importe un composant `UserProfileCard` et lui définit des composants enfants.

```
import UserProfileCard from './UserProfileCard.jsx';  
import Avatar from './Avatar.jsx';
```

```
export default function SettingsPage() {  
  return (  
    <UserProfileCard>  
      <Avatar />  
      <p>Bienvenue sur votre profil !</p>  
    </UserProfileCard>  
  );  
}
```

```
export default function UserProfileCard({ children }) {  
  return (  
    <div>  
      {children}  
    </div>  
  );  
}
```

Le composant `UserProfileCard` reçoit tout ce qui a été passé entre les balises du composant grâce à la props `children`.