Chapitre 5 : Les composants et props

Table des matières

I. Définition

Nous utilisons les props dans React pour transmettre des données d'un composant à un autre (d'un composant parent à un ou plusieurs composants enfants). Props est juste une façon plus courte de dire propriétés. Ils sont utiles lorsque vous souhaitez que le flux de données dans votre application soit dynamique.

II. Affectation par décompositions

L'**affectation par décomposition** (*destructuring* en anglais) est une expression JavaScript qui permet d'extraire (*unpack* en anglais) des données d'un tableau ou d'un objet grâce à une syntaxe dont la forme ressemble à la structure du tableau ou de l'objet.

Exemple 1 :

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]
```

Exemple 2 :

```
const toto = ["un", "deux", "trois"];

// sans utiliser la décomposition
const un    = toto[0];
const deux  = toto[1];
const trois = toto[2];

// en utilisant la décomposition
const [un, deux, trois] = toto;
```

Exemple 3 : Echange de valeurs

```
let a = 1;
let b = 3;

[a, b] = [b, a];
console.log(a); // 3
console.log(b); // 1
```

Exemple 4 : Utilisation d'une fonction

```
function f() {
  return [1, 2];
}

[a,b] = f()
```

Exemple 5 : Ignorer des valeurs

```
function f() {
  return [1, 2, 3];
}

const [a, , b] = f();
console.log("A vaut " + a + " B vaut " + b);
```

Exemple 6 : Décomposer un tableau

```
const [a, ...b] = [1, 2, 3];
console.log(a); // 1
console.log(b); // [2, 3]
```

Exemple 7 : Décomposer un objet

```
const o = {p: 42, q: true};
const {p, q} = o;

console.log(p); // 42
console.log(q); // true

// Assign new variable names
const {p: toto, q: truc} = o;
```

```
console.log(toto); // 42
console.log(truc); // true
```

III. Composant fonctionnel et props

Les props React sont comme des arguments de fonction en JavaScript et des attributs en HTML.

Pour envoyer des props dans un composant, utilisez la même syntaxe que les attributs HTML. Comme le montre l'exemple suivant :

```
<Voiture marque="Mercedes" modele="2010"/>
```

Le composant est déclaré de la façon suivante :

```
function Voiture(props) {
  return <h1>La marque : {props.marque} - Le modele : {props.modele}</h1>
}
```

Voici un autre exemple d'utilisation du props :

```
function Car(props) {
  return <h2>I am a {props.brand.model}!</h2>;
}

function Garage() {
  const carInfo = { name: "Ford", model: "Mustang" };
  return (
    <>
      <h1>Who lives in my garage?</h1>
      <Car brand={carInfo} />
    </>
  );
}

function App() {
  return <Garage />;
}

export default App;
```

IV. Composant classe et props

Voici un exemple d'utilisation du props avec un composant sous format classe :

```
import React from "react";

class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand.model}!</h2>;
  }
}
class Garage extends React.Component {
  render() {
    const carInfo = {name: "Ford", model: "Mustang" };
    return (
      <>
        <h1>Who lives in my garage?</h1>
        <Car brand={carInfo} />
      </>
    );
  }
}

function App() {
  return <Garage />;
}

export default App;
```

Voici un autre exemple en utilisant les listes :

```
import React from "react";

class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.brand}!</h2>;
  }
}
class Garage extends React.Component {
  render() {
    const brands = ["Ford", "Mercedes", "Hyundai"];
    return (
      <>
        <h1>Who lives in my garage?</h1>
        <ul>
          <li>
            {brands.map((item) => (
              <Car brand={item} />
```

```
            ))}
          </li>
        </ul>
      </>
    );
  }
}

function App() {
  return <Garage />;
}

export default App;
```

V. Passer dynamiquement un contenu a un composant

Prenons l'exemple suivant :

```
import React from "react";
import ReactDOM from "react-dom/client";
import Presentation from "./components/Presentation";

const element = document.getElementById("root");
const root = ReactDOM.createRoot(element);

function App() {
  return (
    <div>
      <Presentation nom="Rami" prenom="Ahmed">
        <p>ce ci est un children props</p>
      </Presentation>
      <Presentation nom="Kamali" prenom="Ali">
        <button>quitter</button>
      </Presentation>
    </div>
  );
}

export default App
```

Dans ce cas l'élément Présentation contient un élément enfant entre la balise ouvrante et fermante.

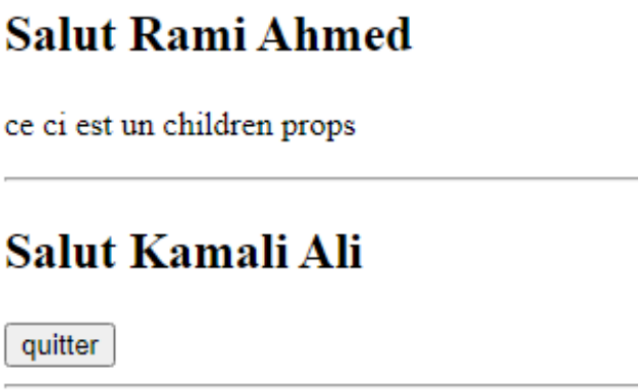
Les éléments enfants sont différents, un paragraphe et un bouton.

```
function Presentation(props) {
```

```
  return (
    <div>
      <h2>
        Salut {props.nom} {props.prenom}
      </h2>
      {props.children}
      <hr />
    </div>
  );
}
```

Pour récupérer l'élément enfant en utilise la propriété children : ***{props.children}***

Le rendu



Voici un autre exemple d'utilisation du props.children :

```
class App extends Component {
  render() {
    return <Box>
             <Text></Text>
           </Box>;
  }
}

class Box extends Component {
  render() {
    return <div>{this.props.children}</div>;
  }
}
class Text extends Component {
  render() {
    return <p>Welcome to App</p>;
  }
}
```