Chapitre 8 : Personnalisés les composants

Table des matières

I. Importer les images

Cela peut se faire de deux manières :

- En utilisant l'instruction import
- En utilisant la fonction require().
- En mettant les images dans le dossier public.

a. L'instruction import

Parce qu'elle est plus facile à lire et à comprendre, l'instruction import est la méthode la plus couramment utilisée pour importer des images stockées localement dans React.

Les images sont traitées comme des exportations par défaut, et lorsque nous les importons, nous le faisons de la même manière que nous importons des composants. Pour ce faire, nous spécifions le chemin relatif du fichier vers l'image que nous importons. Par exemple:

```
import Logo from './images/react-logo.png';

const App = () => {
   return (
      <div>
         <img src={Logo} alt="React Logo" />
      </div>
   );
};
```

b. La fonction require

La fonction require() est une fonction Node.js qui est utilisée pour inclure des modules externes à partir des fichiers autres que le fichier courant. Elle fonctionne de la même manière que l'instruction import et nous permet d'inclure des images.

Exemple 1 :

```
let Logo = require('./images/react-logo.png');

const App = () => {
   return (
      <div>
         <img src={Logo} alt="React Logo" />
      </div>
```

```
    );
  };
```

Exemple 2 :

```
const App = () => {
   return (
      <div>
         <img src={require('./images/react-logo.png')} alt="React
Logo" />
      </div>
   );
};
```

c. Les images dans le dossier public

Si les images sont stockés dans le dossier Public du projet, nous ne nous pouvons pas utiliser les deux méthodes qui nous avons vu précédemment.

Pour importer une image stocker dans le dossier public, nous utilisons le code suivant :

```
   <img src={process.env.PUBLIC_URL+"/images/face1.png"}/>
```

II. Styliser les composants React

a. Inline CSS

Pour donner du style à un élément avec l'attribut en ligne **style**, la valeur doit être un objet JavaScript.

Exemple 1 :

```
function App() {
  return (
    <div className="App">
     <button style={{color:whitee, backgroundColor:'blue', border:"none"}} />
    </div>
  );
}
```

Exemple 2 :

```
function App() {
  const style = { color: whitee, backgroundColor: "blue", border: "none" };

  return (
    <div className="App">
      <button style={style} />
    </div>
  );
}
```

b. Modules CSS

Un module CSS est un fichier CSS. Mais avec une différence essentielle : par défaut, lorsqu'il est importé, chaque nom de classe à l'intérieur d'un module CSS a une portée locale pour le composant qui l'importe. Cela vous permet d'utiliser pratiquement n'importe quel nom valide pour vos classes, sans vous soucier des conflits avec d'autres noms de classes dans votre application.

Les avantages d'utiliser CSS module :

- L'utilisation des modules CSS permet d'éviter les conflits d'espace de nom pour les classes CSS. Plusieurs fichiers CSS peuvent contenir la même classe CSS.
- L'un des principaux avantages de l'utilisation des modules CSS est que vous pouvez modifier n'importe quel fichier CSS avec assurance et sans vous soucier de l'impact sur d'autres sites.

La création d'un module CSS pour un composant est relativement facile. Il faut seulement insérer le mot module avant .css, comme ceci : **[nom-file-css].module.css**

Par exemple : Tache.module.css.

```
.myclass {
    color: whitesmoke;
    background-color: red;
}
```

L'exemple suivant l'utilisation d'un CSS module dans un composant :

```
import React, { Component } from 'react'
import styles from './Personne.module.css'
```

```
export default class Personne extends Component {
  render() {
    return (
      <div className={styles.myclass}>Personne</div>
    )
  }
}
```

c. Composants stylisés

Styled-components est une bibliothèque construite pour les développeurs React et React Native.

Elle vous permet d'utiliser des styles au niveau des composants dans vos applications. Il va nous permettre de créer des composants personnalisés en écrivant des CSS réels dans votre JavaScript.

Installation de styled-component :

L'installation de la bibliothèque **styled-component** se fait à l'aide de la commande suivante :

```
npm i --save styled-components
```

Voici la syntaxe de base de declaration d'un composant stylise.

```
import styled from "styled-component";

const ComponentName = styled.DOMElementTag`
    cssProperty: cssValue
`
```

Voici quelques exemples de déclaration des composants stylises :

Exemple 1:

```
import styled from "styled-components";

const ButtonStyles = styled.button`
    font-family: "Poppins", sans-serif;
    border: none;
    border-radius: 0;
    padding-inline: 1.75rem;
```

```
    padding-block: .75rem;
    background-color: springGreen;
    color: white;
`;

const StyledButton = () => {
  return (
    <ButtonStyles> ## Now, our styles will get applied.
      Hello World!
    </ButtonStyles>
  )
}
```

Exemple 2 :

```
import styled from "styled-components";

const ButtonStyles = styled.button`
background: black;
color: white;
border-radius: 7px;
padding: 20px;
margin: 10px;
font-size: 16px;
:disabled {
  opacity: 0.4;
}
:hover {
  box-shadow: 0 0 10px yellow;
}
`;

const StyledButton = () => {
  return (
    <ButtonStyles> ## Now, our styles will get applied.
      Hello World!
    </ButtonStyles>
  )
}
```

Exemple 3 :

```
import styled from 'styled-components'

const UnorderedList = styled.ul`
  list-style: none;
```

```
  li {
    margin-bottom: 7px;
  }
`

export default App = () => (
    <UnorderedList>
        <li>...</li>
    </UnorderedList>
)
```

Exemple 4 :

```
const StyledButton = styled.button `
    font-size: 30px;
    border-raduis: 10px;
    border: none;
    color: white;
    background-color: ${props => props.bg};
    &:hover {
        opacity: 0.7;
    }
`;

export default function ExempleStyledComponent() {
  return (
    <div>
        <StyledButton bg='red' >Cliquer ici</StyledButton>
    </div>
  )
}
```

III. Affichage conditionnel

Dans React, nous pouvons créer plusieurs composants qui encapsulent le comportement dont nous avons besoin. Ensuite, nous pouvons les rendre en fonction de certaines conditions ou de l'état de notre application. En d'autres termes, en fonction d'une ou plusieurs conditions, un composant décide des éléments qu'il va retourner.

Dans React, le rendu conditionnel fonctionne de la même manière que les conditions en JavaScript. Nous utilisons des opérateurs JavaScript pour créer des éléments représentant l'état actuel, puis le composant React met à jour l'interface utilisateur pour les faire correspondre.

Il y a plus d'une façon de faire du rendu conditionnel dans React. Elles sont données ci-dessous.

- L'instruction if
- L'opérateur ternaire
- L'opérateur logique &&
- L'opérateur switch case

a. L'instruction if

C'est le moyen le plus simple d'avoir un rendu conditionnel dans React. Voici un exemple avec cette instruction :

```
function UserLoggin(props) {
  return <h1>Welcome back!</h1>;
}

function GuestLoggin(props) {
  return <h1>Please sign up.</h1>;
}

function SignUp(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserLoggin />;
  }
  return <GuestLoggin />;
}

function App() {
  return <SignUp isLoggedIn={false} />;
}
```

b. L'opérateur ternaire

L'opérateur ternaire est utilisé dans les cas où deux blocs s'alternent sous une certaine condition. Cet opérateur rend votre instruction if-else plus concise. Il prend trois opérandes et est utilisé comme un raccourci pour l'instruction if.

```
function Example()
{
  const isLoggedIn = this.state.isLoggedIn;
```

```
  return (
    <div>
      Welcome {isLoggedIn ? 'Back' : 'Please login first'}.
    </div>
  );
}
```

c. L'opérateur logique &&

Cet opérateur est utilisé pour vérifier la condition. Si la condition est vraie, il retournera l'élément juste après &&, et si elle est fausse, React l'ignorera et le sautera. Voici un exemple d'utilisation de cette operateur :

```
function Example()
{
    return(<div>
            {
                (10 > 5) && alert('This alert will be shown!')
            }
           </div>
        );
}
```

d. L'opérateur switch case

Il est parfois possible d'avoir plusieurs rendus conditionnels. Dans le cas du switch, le rendu conditionnel est appliqué sur la base d'un état différent.

```
function NotificationMsg({ text}) {
  switch(text) {
    case 'Hi All':
      return <Message: text={text} />;
    case 'Hello JavaTpoint':
      return <Message text={text} />;
    default:
      return null;
  }
}
```

IV. Les listes et clés

a. Les listes

Les listes sont utilisées pour afficher des données dans un format ordonné et sont principalement utilisées pour afficher des menus sur les sites Web. Dans React, les listes peuvent être créées de la même manière que les listes en JavaScript.

b. 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.

V. Communication inter-composants

a. Du parent a l'enfant

La direction la plus simple du flux de données est la descente dans la hiérarchie, du parent à l'enfant. Le mécanisme de React pour accomplir ceci est appelé props. Un composant React est une fonction qui reçoit un paramètre appelé props.

b. De l'enfant au parent avec des callbacks

Un Callback est une fonction qui est appelé comme argument dans une autre fonction.

Comme nous l'avons appris, les parents transmettent des données aux enfants par le biais de props.

Un "spécial" props de type fonction peut être transmis à un enfant. Au moment d'un événement (par exemple, une interaction avec l'utilisateur), l'enfant peut alors appeler cette fonction en tant que callback. Voici un exemple :

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

function BookTitle(props) {
  return (
    <label>
      Title:
      <input onChange={props.onTitleChange} value={props.title} />
```

```
    </label>
  )
}


class BookEditForm extends Component {
  constructor(props){
    super(props)

    this.state =
      {
        title : this.props.book.title
      }

      this.handleTitleChange = this.handleTitleChange.bind(this);
  }
  handleTitleChange(evt) {
    this.setState({title: evt.target.value})
  }
  render() {
    return (
      <form>
        <BookTitle onTitleChange={this.handleTitleChange}
title={this.state.title} />
      </form>
    )
  }
}



function App() {
  return <BookEditForm book={{title:'Les miserable'}}/>;
}
```