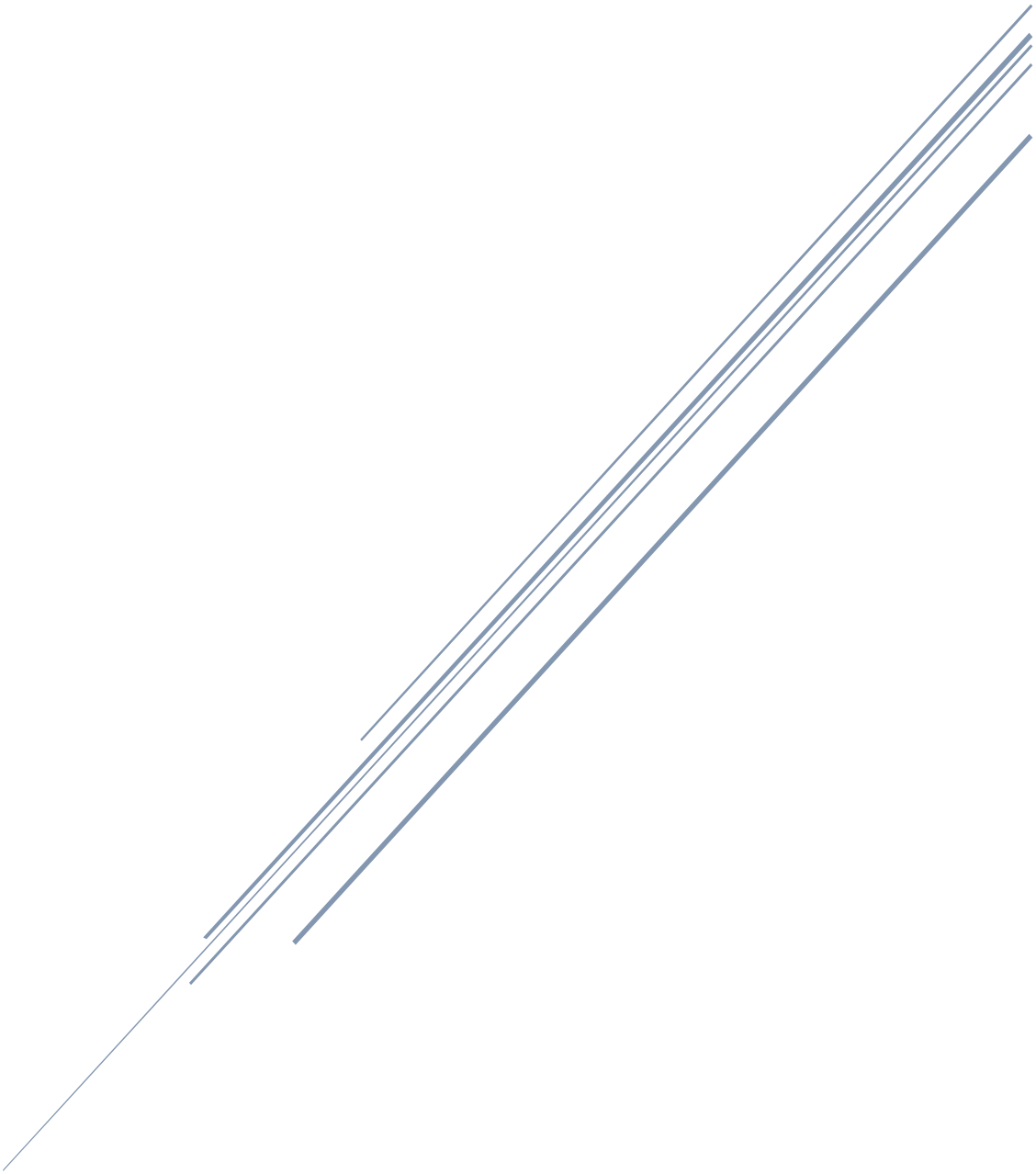
PERSONNALISER LES COMPOSANTS REACT

Gestion des images, la stylisation des composants, l'affichage conditionnel, et la communication inter-composants.



ISMO

Chapitre 08 – Personnaliser les composants React

MAHDI KELLOUCH

Table des matières

I. Importer les images

a. Utilisation de l'instruction import

L'instruction **import** est la méthode la plus courante et la plus lisible pour importer des images locales dans un projet React.

Exemple :

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

b. Utilisation de la fonction require()

La fonction **require()** est une méthode héritée de Node.js qui permet d'inclure des fichiers dans votre code. Elle fonctionne également avec les images.

Exemple 1 :

```
const Logo = require('./images/react-logo.png');
// const LogoSVG = require('./images/react-logo.svg').default;
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. Images dans le dossier public

Lorsque les images sont placées dans le dossier public, l'accès se fait en utilisant **process.env.PUBLIC_URL**.

Exemple :

```
<img src={process.env.PUBLIC_URL + "/images/ react-logo.png"} alt="Face" />
```

Exercice d'application - Importation d'images

1. Créez un projet **React** et ajoutez une image dans le dossier **src/images/.** Utilisez import pour l'afficher dans votre application.
2. Répétez l'exercice en utilisant **require()** et en stockant l'image dans le dossier public.

II. Styliser les composants React

a. Inline CSS

L'inline CSS permet d'ajouter des styles directement dans le JSX à l'aide de l'attribut style. Les valeurs doivent être des objets JavaScript.

Exemple 1 :

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

Exemple 2 :

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

b. Modules CSS

Les modules CSS permettent d'éviter les conflits de noms de classes en encapsulant les styles dans un espace local.

Exemple :

1. Créez un fichier Personne.module.css :

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

2. Utilisation 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>
    );
  }
}
```

Les avantages de cette approche avec CSS Modules :

1. **Portée locale** : Les styles sont scoped à chaque composant, évitant les conflits de noms
2. **Classes uniques** : Les noms de classes sont automatiquement générés avec un hash unique
3. **Maintenabilité** : Organisation claire avec un fichier CSS dédié par composant
4. **Réutilisabilité** : Chaque composant est autonome avec ses propres styles
5. **Débogage facilité** : Les styles sont isolés, rendant plus simple la détection des problèmes

c. Styled-components

styled-components permet d'écrire du CSS au sein même de vos composants React. Ce style est lié au composant et ne risque pas de provoquer des conflits.

Installation :

```
npm install styled-components
```

Exemple 1 :

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

const Button = styled.button`
  font-family: "Poppins", sans-serif;
  background-color: springGreen;
  color: white;
```

```
  padding: 1rem 1.75rem;
  border: none;
  border-radius: 0;
`;

const App = () => {
  return (
    <Button>Hello World!</Button>
  );
}
```

Exemple 2 :

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

const Button = styled.button`
  background: black;
  color: white;
  border-radius: 7px;
  padding: 20px;
  &:hover {
    box-shadow: 0 0 10px yellow;
  }
`;

const App = () => {
  return (
    <Button>Click Me</Button>
  );
};
```

Exercice d'application - Stylisation

1. Créez un bouton stylisé en utilisant un composant styled-components.
2. Utilisez des modules CSS pour styliser un composant dans votre application.

III. Affichage conditionnel dans React

a. L'instruction if

L'instruction if permet de rendre un composant en fonction d'une condition.

Exemple :

```
function SignUp(props) {
    const isLoggedIn = props.isLoggedIn;
    if (isLoggedIn) {
      return <h1>Welcome back!</h1>;
```

```
    }
    return <h1>Please sign up.</h1>;
  }
```

b. L'opérateur ternaire

L'opérateur ternaire permet d'écrire des conditions plus compactes.

Exemple :

```
function Example() {
    const isLoggedIn = this.state.isLoggedIn;
    return (
      <div>
        Welcome {isLoggedIn ? 'Back' : 'Please login first'}.
      </div>
    );
  }
```

c. L'opérateur logique &&

Utilisez **&&** pour conditionner l'affichage d'un élément.

Exemple :

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

d. L'opérateur switch

Le switch est utilisé lorsque plusieurs conditions doivent être testées.

Exemple :

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

Exercice d'application - Affichage conditionnel

1. Créez un composant qui affiche un message de bienvenue si l'utilisateur est connecté et un message d'invitation à se connecter sinon.
2. Utilisez l'opérateur ternaire pour afficher un message en fonction de l'état de l'utilisateur.

IV. Communication inter-composants

a. Du parent à l'enfant

Le flux de données dans React suit le modèle parent-enfant. Le parent passe des données aux enfants via les props.

Exemple :

```
function Child(props) {
    return <div>{props.message}</div>;
  }

  function Parent() {
    return <Child message="Hello from parent!" />;
  }
```

b. De l'enfant au parent avec des callbacks

L'enfant peut envoyer des données au parent via un callback passé en props.

Exemple :

```
 function Child(props) {
    return <button onClick={props.onClick}>Click me</button>;
  }

  class Parent extends React.Component {
    handleClick = () => {
      alert('Button clicked!');
    };

    render() {
      return <Child onClick={this.handleClick} />;
    }
  }
```

Exercice d'application - Communication inter-composants

1. Créez un composant parent qui passe une donnée à un composant enfant via props.
2. Implémentez une fonction callback dans l'enfant pour modifier l'état du parent.