Développement front

MMI 2 – TP#2 S4

IUT CERGY-PONTOISE

Danielo **JEAN-LOUIS**
Michele **LINARDI**

Props et State

- Permettent de changer / initialiser le contenu d'un composant React

Props

- Immutable et en lecture seule
  - On ne peut pas modifier une props
- Permet la communication entre composants
  - Passé par le parent le plus proche

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Props

- Un composant peut avoir un nombre infini de props
- Premier paramètre d'un composant
  - S'exprime sous forme d'objet
  - Tous les composants React ont ce paramètre

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Props

```
const MonComposant = (props) => {
    return (
        <p>Ma matière préférée est {props.matiere}.</p>
    )
}
```

Ici notre composant "MonComposant" attend une props appelée "matière"

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Props

On définit notre composant "MonComposant" avec 2 props :
- matière = "anglais"
- age = 14

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Props

- N'acceptent que des primitives comme valeurs :
  - Entiers (négatif ou non)
  - Chaîne de caractères
  - Tableau
  - Objet
  - Booléen
  - Fonction

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Props

- Entier (négatif ou positif), tableau, objet et booléen doivent être mis entre accolades pour être gérés
- L'ordre des props n'a aucune importance

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html

Pratiquons ! - Découvrons ReactJS (Partie 1)

Pré-requis :

- Avoir la ressource ressources/props-et-state

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-2/ressources

props.children

- Props propre à tous les composants react
- Permet d'afficher les enfants d'un composants

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html
- https://fr.reactjs.org/docs/composition-vs-inheritance.html

props.children

```jsx
const MyComp = (props) => {
    return (
        <section>
            <h1>Hello</h1>
            {props.children}
        </section>
    )
}

const MyApp = () => {
    return (
        <MyComp>
            <p>Hello world</p>
        </MyComp>
    )
}
```

Ici notre composant <MyComp> va afficher ce qui a été mis dans sa balise grâce à la props "children"

Source(s) :

- https://fr.reactjs.org/docs/components-and-props.html
- https://fr.reactjs.org/docs/composition-vs-inheritance.html

Évènements

- Attribut placé au niveau d'une balise HTML
  - Peut être transmis entre composants
- Ajoute de l'interaction
- Attend comme valeur le nom de la fonction ou une fonction anonyme

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/Events
- https://fr.reactjs.org/docs/handling-events.html

Évènements

```
function MyButton() {
    const handleClick = () => {
        console.log("click")
    }

    return <button onClick={handleClick}>Activer clic</button>;
}
```

Ici, on a placé un évènement "onClick" sur notre balise <button>

Source(s) :

- https://developer.mozilla.org/fr/docs/Web/Events
- https://fr.reactjs.org/docs/handling-events.html

Évènements

- S'écrit en camelCase
  - onclick (html) → onClick (javascript)
- Les règles vues en javascript concernant les évènements et l'accessibilité s'appliquent aussi ici

Source(s) :
- https://developer.mozilla.org/fr/docs/Web/Events
- https://fr.reactjs.org/docs/handling-events.html

Pratiquons ! - Découvrons ReactJS (Partie 2)

Pré-requis :

- Avoir la ressource ressources/props-et-state

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-2/ressources

State

- Mutable, lecture et écriture
  - On peut modifier la valeur d'un state
- Met à jour l'application à chaque mise à jour
- Géré au sein même d'un composant
- Peut être initialisé par une props
- Un composant peut avoir plusieurs states

Source(s) :

- https://fr.reactjs.org/docs/hooks-state.html

State

```jsx
function MonCompteur() {
  const [count, setCount] = React.useState(0);

  return (
      <button onClick={() => setCount(count + 1)}>
        Vous avez cliqué {count} fois
      </button>
  );
}
```

Source(s) :

- https://fr.reactjs.org/docs/hooks-state.html

State – Analysons ça ensemble

```
const [count, setCount] = React.useState(0);
```

- Définition de notre state
  - [count,] : getter. Permet de récupérer la valeur
  - [, setCount] : setter. Permet de modifier la valeur
  - React.useState() : Initialisation du state

Source(s) :

- https://fr.reactjs.org/docs/hooks-state.html

State – Analysons ça ensemble

```
<button onClick={() => setCount(count + 1)}>
```

- Modification du state
  - Ici on prend la valeur actuelle de count et on l'incrémente de 1 à chaque clic

Source(s) :
- https://fr.reactjs.org/docs/hooks-state.html

State – Analysons ça ensemble

```
Vous avez cliqué {count} fois
```

- Affichage de la valeur du state

Source(s) :
- https://fr.reactjs.org/docs/hooks-state.html

Hooks

- Syntaxe moderne des composants (depuis la version 16.8)
- Allègent et simplifient le code
- Remplacent l'utilisation de classes

Source(s) :

- https://fr.reactjs.org/docs/hooks-intro.html

Hooks

```
class MyFirstComponent extends React.Component {
    constructor(props) {
        this.state = {
            message: "Bonjour tout le monde"
        }
    }

    render() {
        return (
            <button onClick={() => {
                this.setState({
                    message: "Au revoir tout le monde"
                })
            }}>
                {this.state.message}
            </button>
        );
    }
}
```

```
const MyFirstComponent = () => {
    const [message, setMessage] = "Bonjour tout le monde";

    return (
        <button onClick={() => {
            setMessage("Au revoir tout le monde")
        }}>
            {message}
        </button>
    );
}
```

A gauche une classe React (ancienne syntaxe), à droite un composant fonctionnel

Hooks

- Possibilité de créer ses propres hooks
- Existe de multiples hooks natifs
  - React.useState(), React.useEffect()
- **Un hook doit toujours être à la racine d'un composant**

Source(s) :

- https://fr.reactjs.org/docs/hooks-intro.html

Hooks

```
if (name !== '') {
    React.useEffect(() => {
        // Mon code
    });
}
```

Interdit

Notre hook est dans une condition.
Donc pas à la racine.

```
React.useEffect(() => {
    if (name !== '') {
        // Mon code
    }
});
```

Correct

Notre hook est à la racine.

Source(s) :

- https://fr.reactjs.org/docs/hooks-intro.html
- https://fr.reactjs.org/docs/hooks-rules.html

Pratiquons ! - Découvrons ReactJS (Partie 3/4)

Pré-requis :

- Avoir la ressource ressources/props-et-state

A télécharger ici :
https://download-directory.github.io?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-2/ressources

State

- Un state ne peut pas être modifié par un composant externe
- Par convention, le setter d'un state commence par "set"
  - Exemples : setName, setListItems...

Source(s) :
- https://fr.reactjs.org/docs/hooks-state.html

Questions ?