

Développement front

MMI 2 – TP#2 S4

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

Props et State

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

Props

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

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.react.dev/learn/passing-props-to-a-component>

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.react.dev/learn/passing-props-to-a-component>

Props

```
const FicheEtudiantComplete = () => {  
  return (  
    <section>  
      <MonComposant matiere="Anglais" age={14} />  
    </section>  
  )  
}
```

Props 1

Props 2

On définit notre composant "MonComposant" avec deux props :

- **matière = "anglais"**
- **age = 14**

Source(s) :

- <https://fr.react.dev/learn/passing-props-to-a-component>

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, variables doivent être mis entre accolades pour être transmis
- 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/DanYeellow/cours/tree/main/developpement-front-s4/travaux-pratiques/numero-2/ressources>

Évènements

- Ajoutent de l'interaction
- Peuvent être transmis entre composants
- Attendent comme valeur le pointeur de fonction ou une fonction anonyme

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/Events>
- <https://fr.react.dev/learn/responding-to-events>

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.react.dev/learn/responding-to-events>

Évènements

```
function MyButton() {  
  const handleClick = () => {  
    console.log("click")  
  }  
  
  return <button onClick={handleClick}>Activer clic</button>;  
}
```

...et on lui passe le pointeur de fonction "handleClick"

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/Events>
- <https://fr.react.dev/learn/responding-to-events>

Évènements

- S'écrivent en camelCase
 - onclick (html) → onClick (javascript)
- Les règles vues en javascript concernant les évènements et l'accessibilité s'appliquent aussi ici
 - Ex : pas de onClick sur un <a>

Source(s) :

- <https://developer.mozilla.org/fr/docs/Web/Events>
- <https://fr.react.dev/learn/responding-to-events>

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/DanYeellow/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.react.dev/learn/managing-state>

State

```
function MonCompteur() {  
  const [count, setCount] = React.useState(0);  
  
  return (  
    <button onClick={() => setCount(count + 1)}>  
      Vous avez cliqué {count} fois  
    </button>  
  );  
}
```

Source(s) :

- <https://fr.react.dev/learn/managing-state>

State – Analysons ça ensemble

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

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

Source(s) :

- <https://fr.react.dev/learn/managing-state>

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.react.dev/learn/managing-state>

State – Analysons ça ensemble

`Vous avez cliqué {count} fois`

- Affichage de la valeur du state (dans le return du composant)

Source(s) :

- <https://fr.react.dev/learn/managing-state>

Hooks

- Syntaxe moderne des composants (depuis la version 16.8 de React)
- Allègent et simplifient le code
- Remplacent l'utilisation de classes
- Permet la création de composants dits “fonctionnels”

Source(s) :

- <https://fr.react.dev/reference/react/hooks>

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] = React.useState(  
    "Bonjour yout 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()`
- **Les hooks sont soumis à des règles**
 - Voir slide suivante

Source(s) :

- <https://fr.react.dev/reference/react/hooks>

Hooks

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



Interdit

Notre hook "useState" est dans une condition

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



Correct

Notre hook "useState" est à la racine

Source(s) :

- <https://fr.react.dev/reference/react/hooks>
- <https://fr.react.dev/warnings/invalid-hook-call-warning>

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/DanYeellow/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.react.dev/reference/react/useState>

Questions ?

