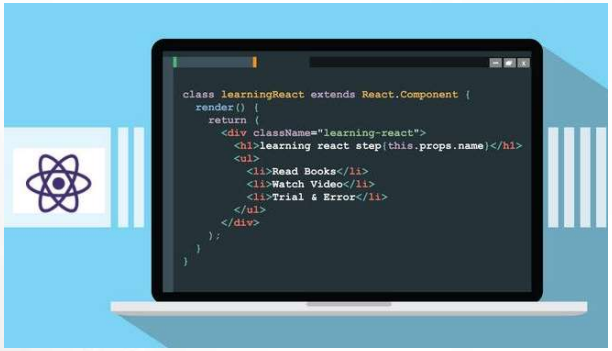
Chapitre 5 : Les props



Intérêt

- Une des clés de React est la notion que les données « descendent » à travers l'arborescence de composants.
- Le mécanisme-clé pour cela ce sont les props est l'abréviation de "properties".
- Avec React on qualifie de composant enfant tout composant défini dans le render() du composant parent, quel que soit son niveau de profondeur.
- On dit donc que le composant qui fournit le render() est le parent, et que tout composant figurant dans ce render() est un enfant.

- Deux règles sont importantes :

  - Une prop est toujours passée par un composant parent à ses composants enfants : c'est le seul moyen normal de transmission

  - Une prop est considérée en lecture seule dans le composant qui la reçoit

Présentation

- La plupart des composants peuvent être personnalisés avec différents props lors de leur création.
- Les props sont des arguments transmis aux composants React.
- Ils sont transmis aux composants via des attributs HTML.
- Ils sont semblables à des arguments de fonction en JavaScript et des attributs en HTML.
- C’est pour cela que pour envoyer des props dans un composant, on utilise la même syntaxe que les attributs HTML.

Props techniques

- Key :
  - Lorsqu'on manipule des tableaux dans une grappe React, il est impératif d'équiper chaque élément du tableau d'une prop spéciale `key` . Cela permettra à React de gérer au mieux l'évolution du tableau d'un `render()` à l'autre.
  - L'absence de cette prop entraîne un avertissement en mode développement. Par ailleurs, elle n'est pas consultable par le composant enfant qui la reçoit : elle ne figure pas dans sa liste de props.
- Children :
  - Cette prop est particulière : elle n'est pas fournie à l'aide d'un attribut, mais en imbriquant des composants à l'intérieur du composant concerné.
  - La liste des composants imbriqués constitue alors la prop `children` du composant qui les « entoure ». Elle est donc automatiquement renseignée.
  - Exemple : La prop `children` du composant `<FileList />` est un tableau contenant `<UploadCreator />` et `<StatusBar />`

```
return ( <FileList> <UploadCreator /> <StatusBar /> </FileList> )
```

Props vs State

Conditions	Props	State
Reçoit la valeur initiale à partir du composant parent	✓	✓
Le composant parent peut changer la valeur	✓	✗
Met des valeurs par défaut dans le composant	✓	✓
Sera modifié dans le composant	✗	✓
Met des valeurs par défaut pour les composants children	✓	✓
Sera modifié dans le composant children	✓	✗

Valeurs par défaut

- Un composant peut définir des valeurs par défaut pour ses props.
- Pour cela, la définition de composants doit être effectuée à l'aide de fonctions.
- Exemple :

```
const Welcome = (props) => {
  return <h1>Hello, {props.nom}</h1>;
}
```

Mécanisme de props dans les classes 1/3

- Les props d'un composant sont sous forme d'un objet qui contient des informations sur ce composant.
- Pour voir les props de ce dernier on utilise l'expression : this.props.
- En règle générale, les props sont immuables, donc ne doivent pas être changés.
- C'est que les données en React circulent en une seule direction : du parent vers l'enfant.
- Ainsi, les props ne doivent pas être modifiés à l'intérieur des composants puisque ces derniers reçoivent des props de leur parent.

Mécanisme de props dans les classes 2/3

Exemple :

- Le composant reçoit l'argument en tant qu'objet props (ici c'est text) :
- *Composant Hello.js*

```
import React from 'react';
class Hello extends React.Component {
    render() {
       return(
          <div>
              <h1>{this.props.text}</h1>
         </div>
      );
   }
}
export default Hello;
```

Mécanisme de props dans les classes 3/3

- Les props sont également la façon dont on transmet les données d'un composant à un autre, en tant que paramètres. On envoie donc la propriété "text" du composant App au composant Hello :

- *Composant App.js*

```
import React from 'react';
import ReactDOM from "react-dom";
import Hello from './Hello';
class App extends React.Component {
render() {    return (
                  <div>
                     <Hello text="Bonjour"/>
                  </div>    );
         }  }
const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
export default App;
```

Envoi de variable

- Si on a une variable à envoyer, et non une chaîne comme dans l'exemple ci-dessus, on met simplement le nom de la variable entre accolades :

- *Composant App.js*

```
import React from 'react';
import Hello from './Hello';
class App extends React.Component {
render() {    const message = "Bonjour";
  return (
     <div>
        <Hello text={message}/>
     </div>
  );  }   }
export default App;
```

Envoi d'un objet 1/2

- Dans le cas où il s'agit d'un objet, on crée un objet nommé "messages" :
- *Composant App.js*

```
import React from 'react';
import ReactDOM from "react-dom";
import Hello from './Hello';
class App extends React.Component {
render() {  const messages = {sujet: "Bonjour", contenu: "Salutations"};
           return ( <div>
                      <Hello text={messages}/>
                    </div>
           );  }   }
export default App;
```

Envoi d'un objet 2/2

- Puis on l'envoie au composant Hello :
- *Composant Hello.js*

```
import React from 'react';

class Hello extends React.Component {
    render() {
       return(
          <div>
              <h1> {this.props.text.contenu} </h1>

          </div>
      );
   }
}
export default Hello;
```

Les props dans le constructeur 1/2

- Si le composant a une fonction constructeur, les props doivent toujours être passés au constructeur et également au React.Component via la méthode super ().
- En JavaScript, super fait référence au constructeur de la classe parente. (Dans notre exemple, ça pointe sur l'implémentation de App).

- *Composant Hello.js*

```
import React from 'react';
class Hello extends React.Component {
    constructor(props) {
        super(props);
    }
   render() {   return(
                  <div>
                      <h1>{this.props.text.contenu}</h1>
                  </div>
      );     }  }
export default Hello;
```

Les props dans le constructeur 2/2

- Comme précédemment, le résultat obtenu est le contenu de l'objet messages c'est-à-dire Salutations.

Remarque :

Il est recommandable de toujours appeler super(props), même si ce n'est pas strictement nécessaire car ça garantit que this.props est défini même avant la fin du constructeur.

Du composant child vers le composant parent 1/3

- Ci-dessous un exemple pour passer les valeurs du composant enfant au composant parent dans reactjs.

```
import React from "react";
class App extends React.Component {
constructor(props) {
super(props);
this.state = {
data: "Initial data..."
}
this.updateState = this.updateState.bind(this);
};
updateState() {
this.setState({data: "Data updated from the child component..."})
}
```

Du composant child vers le composant parent 2/3

```
render() {
return (
<div>
<Content myDataProp = {this.state.data} updateStateProp = {this.updateState} />
</div>
);
}
}
```

Du composant child vers le composant parent 3/3

```
class Content extends React.Component {
render() {
return (
<div>
<button onClick = {this.props.updateStateProp}>CLICK</button>
<h3>{this.props.myDataProp}</h3>
</div>
);
}
}
export default App;
```