ECE PARIS
ÉCOLE D'INGÉNIEURS

Composant: formulaire, communication et stockage

2018/2019

APERCU

Dans ce TP, vous découvrirez comment utiliser des formulaires, communiquer entre des composants ainsi que l'utilisation du stockage local et du stockage de session.

OBJECTIFS

1. Utiliser un formulaire
2. Communiquer avec un composant parent et un composant frère
3. Stocker des informations pendant l'exécution de l'application

Dans ce TP, vous mettrez en place une interface de tchat.

1. Utiliser un formulaire

Tout d'abord, créez un dossier **Chat** à la racine du projet (**src/**) avec un fichier **Chat.js** à l'intérieur. Vous créerez dans **Chat.js** un composant nommé **Chat**. Ce composant contiendra: un élément permettant d'afficher du texte, un champ pour entrer du texte et un bouton d'envoi:

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

class Chat extends Component {
  render() {
    return (
      <div>
        <div class="display"></div>
        <div>
          <input type="text" name="chat"/>
          <button>Envoyer</button>
        </div>
      </div>
    );
  }
}
export default Chat;
```

Maintenant, il faut acquérir la valeur du champ de texte, puis l'afficher dans le conteneur quand le bouton **Envoyer** est cliqué. Pour l'acquisition de la valeur du champ, vous utiliserez l'événement **onChange** pour enregistrer la valeur du champ dès qu'elle est modifiée:

```
handleChange(event) {
  this.setState({ [event.target.name]: event.target.value });
}
```

```
<input type="text" name="chat" onChange={this.handleChange}
value={this.state.chat} />
```

Remarque: event.target.name contient le nom du champ et **event.target.value** contient la valeur du champ. Cela permet de créer une méthode réutilisable pour acquérir les valeurs de n'importe quel champ.

Il faut désormais afficher la valeur dans le conteneur lorsque l'on appuie sur le bouton **Envoyer**. Pour cela, vous utiliserez la propriété **onClick** avec une méthode qui ajoutera le texte dans une variable d'état affichée dans le conteneur.

```
handleSend(event) {
  this.setState({
    display: this.state.display.concat(this.state.chat),
    chat: ''
  });
}

displayChat() {
  let chat = this.state.display.map((item) => {
    return (<p>{item}</p>);
  });

  return (<div class="display">{chat}</div>);
}

render() {
  return (
    <div>
      {this.displayChat()}
      <div>
        <input type="text" name="chat" onChange={this.handleChange}
value={this.state.chat} />
        <button onClick={this.handleSend}>Envoyer</button>
      </div>
    </div>
  );
}
```

2. Communiquer avec un composant

Vous allez désormais avoir au moins deux composants **Chat** dans votre composant principal **App**. L'objectif sera de faire communiquer ces deux chats entre eux.

2.1 Composant parent

Tout d'abord, pour faire communiquer différent composant entre eux, il faut toujours faire remonter l'information au composant commun (le parent dans le cas de composants frère). Pour cela, il faut réussir à faire remonter l'information. Cela se fait au moyen de callback, qui sera passé grâce à une propriété. Vous allez créer une méthode dans le composant parent (**App**) qui permettra de faire remonter l'information.

```
handleSend(name, text) {
  this.setState({
    chat: this.state.chat.concat(`${name}: ${text}`)
  });
}

render() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <h1 className="App-title">Welcome to React</h1>
      </header>
      <Chat onSend={this.handleSend} name="Chat1" />
      <Chat onSend={this.handleSend} name="Chat2" />
    </div>
  );
}
```

Il faut donc ensuite changer la méthode **handleSend** dans le composant **Chat** afin qu'il utilise la propriété **onSend** fourni:

```
handleSend(event) {
  this.props.onSend(this.props.name, this.state.chat);

  this.setState({
    chat: ''
  });
}
```

2.2 Composant frère

Maintenant que le composant **Chat** communique avec le parent, il faut faire descendre l'information. Pour cela, il suffit d'utiliser les propriétés (**props**). La variable d'état de **App** contenant le texte du tchat sera diffusée grâce aux **props** du composant **Chat**.

```
<Chat onSend={this.handleSend} name="Chat1" display={this.state.chat} />
<Chat onSend={this.handleSend} name="Chat2" display={this.state.chat} />
```

Il faut ensuite modifier la méthode **displayChat** du composant **Chat**:

```
displayChat() {
  let chat = this.props.display.map((item) => {
    return (<p>{item}</p>);
  });

  return (<div class="display">{chat}</div>);
}
```

3. Stocker des informations pendant l'exécution de l'application

Il existe aussi d'autres moyens de faire passer de l'information et surtout de permettre de la stocker, bien que cela soit beaucoup moins performant car ce ne sont pas des fonctions liées à React mais à JavaScript.

3.1 Stockage de session

Il est possible de stocker des données liées à un session grâce à la propriété **sessionStorage**. Elle permet d'utiliser un objet **Storage** valable pour la session de navigation en cours et pour les pages du même domaine que la page actuelle. Dans l'objet global **sessionStorage**, les données enregistrées ont une durée de vie limitée et expirent à la fin de la session de navigation actuelle. Une session de navigation dure aussi longtemps que le navigateur est ouvert et s'étend sur plusieurs chargements, rechargements et restaurations de pages. En revanche, une session de navigation n'est valable que pour le contexte de navigation actuel, c'est-à-dire que le fait d'ouvrir une page dans un nouvel onglet ou dans une nouvelle fenêtre provoquera l'initialisation d'une nouvelle session de navigation, ce qui diffère du comportement des sessions utilisant des cookies.

Vous pouvez accéder à l'objet **sessionStorage** n'importe où dans votre application React. Il y a plusieurs méthodes à connaître pour l'utiliser:

```
sessionStorage.setItem("key", "value");
sessionStorage.getItem("key");
sessionStorage.removeItem("key");
```

3.2 Stockage local

La propriété **localStorage** vous permet d'accéder à un objet local **Storage**. Le **localStorage** est similaire au **sessionStorage**. La seule différence: les données stockées dans le **localStorage** n'ont pas de délai d'expiration car les données sont stockées dans le cache du navigateur. Vous pouvez accéder à l'objet **localStorage** n'importe où dans votre application React. Il y a plusieurs méthodes à connaître pour l'utiliser:

```
localStorage.setItem("key", "value");
localStorage.getItem("key");
localStorage.removeItem("key");
```

Plus d'informations: https://developer.mozilla.org/fr/docs/Web/API/Storage