ReactJS
Maîtriser le framework

Sommaire

- Communication inter-composants avec ReactJS
  - Communication inter-composants
  - Gestion des évènements
  - Autobinding
  - Composants de formulaire
  - Manipulation du DOM
  - Styles
  - Création d'une application Single Page Application (SPA) avec ReactJS
- Echanges avec le serveur
  - Présentation de l'architecture REST
  - Echanges entre l'application React et un serveur via REST
- Routage

Communication inter-composants

Comme déjà vu, un composant parent va envoyer des informations à un composant enfant via des propriétés:

- `<Composant message="Hello" />`

- ```
  class Composant extends React.Component {
    constructor(props) {
      super(props);
    }
    render() {
      return { this.props.message }
    }
  }
  ```
  ```

Communication inter-composants

Un composant enfant va envoyer des informations à un composant parent via des une fonction en tant que props:

```
class Child extends React.Component {
  render() {
    return (
      <button onClick={this.props.clickHandler}>
        Add One More
      </button>
    );
  }
}
```

Communication inter-composants

Comme déjà vu, un composant enfant va envoyer des informations à un composant parent via des une fonction en tant que props:

- `<Child clickHandler={this.outputEvent} />`
-
```
outputEvent() {
   console.log('clic enfant');
 }
```

Dans le constructeur:

```
this.outputEvent = this.outputEvent.bind(this);
```

Gestion des évènements

Un composant peut gérer des évènement. Il faut définir une fonction dans la classe qui sera appelée par le gestionnaire d'évènements:

```
render() {
  return (
    <button onClick={this.handleClick}>
      { this.state.isToggleOn ? 'ON' : 'OFF' }
    </button>
  );
}
```

Gestion des évènements

Un composant peut gérer des évènement. Il faut définir une fonction dans la classe qui sera appelée par le gestionnaire d'évènements:

Dans le constructeur:

```
// Cette liaison est nécéssaire afin de permettre
// l'utilisation de `this` dans la fonction de rappel.
this.handleClick = this.handleClick.bind(this);
```

Gestion des évènements

Un composant peut gérer des évènement. Il faut définir une fonction dans la classe qui sera appelée par le gestionnaire d'évènements:

```
handleClick() {
  this.setState(state => ({
    isToggleOn: !state.isToggleOn
}));
```

Autobinding

A cause des changements en version d'ECMAScript, les composants héritent de React.Component au lieu d'utiliser CreateClass comme en ES5.

Pour maintenir l'accès à la variable this dans les méthodes, il faut leur passer le contexte.

```
this.handleClick = this.handleClick.bind(this);
```

Formulaires

En HTML, les éléments de formulaire tels que <input>, <textarea>, et <select> maintiennent généralement leur propre état et se mettent à jour par rapport aux saisies de l'utilisateur.

En React, l'état modifiable est généralement stocké dans la propriété state des composants et mis à jour uniquement avec setState().

Formulaires

On peut combiner ces deux concepts en utilisant l'état local React comme « source unique de vérité ».

Ainsi le composant React qui affiche le formulaire contrôle aussi son comportement par rapport aux saisies de l'utilisateur.

Un champ de formulaire dont l'état est contrôlé de cette façon par React est appelé un « composant contrôlé ».

Formulaires

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
```

Formulaires

```
render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>Nom: </label>
        <input type="text" value={this.state.value}
               onChange={this.handleChange} />
        <input type="submit" value="Envoyer" />
      </form>
    );
}
```

Formulaires

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

handleSubmit(event) {
  alert('Le nom a été soumis : ` +
        this.state.value);
  event.preventDefault();
}
```

Style

Pour rendre les styles dynamiques il faut les mettre dans une variable et utiliser ensuite l'attribut style:

```
let Footer = {
  footer: {fontSize: '2em', color: 'white', background: 'black'},
  a: {color: 'red'},
  p: {textAlign: 'center'}
};
return (
  <footer style={Footer.footer}>
    <p style={Footer.p}> Copyright &copy; { date } - Moi
      <a style={ Footer.a } href="ml.html">ML</a>
    </p>
  </footer>
)
```

Creation d’une Appli SPA

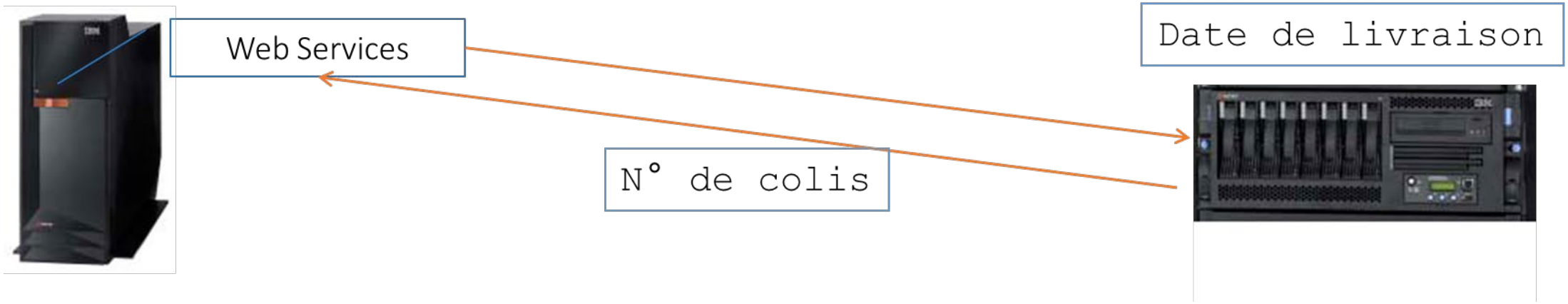
- Créer un nouveau projet Calculatrice.
- Créer une zone pour l’affichage du résultat.
- Créer un formulaire pour la saisie de deux nombres.
- Créer 4 boutons (addition, soustraction, division, multiplication) et les actions correspondantes.

Le projet peut se faire avec un seul composant, ou avec plusieurs composants interagissant entre eux…

Appel d'API REST

Les Web Services (ou API) sont:

- Applications autonomes
- Appelées via Internet (Web)
- Qui exposent une logique métier



Appel d'API REST

Quelques exemples de Web Services:

- Immatriculation des véhicules
- Colissimo
- Google Maps
- Pages jaunes
- Météo
- OpenData

Appel d'API REST

Il existe deux grandes familles d'API :

- Les API SOAP, qui sont essentiellement basés sur le langage XML:
  - Au niveau des avantages, il est ouvert, extensible, décrit la façon dont le client et le serveur doivent communiquer.
  - Au niveau des inconvénients, il est très verbeux.
- Les API REST, qui sont les plus courantes aujourd'hui:
  - Au niveau des avantages, elles sont plus légères et plus flexibles que SOAP.
  - Au niveau des inconvénients, elles peuvent souffrir de problèmes de sécurité et de conformité des transactions.

Appel d’API REST

On peut communiquer avec une API REST:

- Directement avec les fonctions JavaScript fetch().then().then()
- En utilisant une bibliothèque comme Axios.

Appel d'API REST

```
class Yn extends React.Component {
  constructor(props) {
    super(props);
    this.state = { answer: '', img: ''};
  }
  componentDidMount() {
    fetch('http://yesno.wtf/api')
    .then((response) => response.json())
    .then((data) => {
      this.setState(() => ({answer: data.answer, img: data.image}))
    });
  }
  …
}
```

Appel d’API REST

```
render() {
   return <div>
      { this.state.answer }
      <img src={ this.state.img } alt={
this.state.answer } />
   </div>
}
```

Appel d’API REST

La bibliothèque Axios permet de communiquer avec des API selon les méthodes

- GET
- POST
- PUT
- DELETE

```
npm install axios
```

Appel d’API REST

```
axios.get('https://geo.api.gouv.fr/communes', {
  params: {
     nom: 'ecully',
     fields: 'centre,departement’
  }
})
.then((dt) => { console.log(dt); });
```

Routage

Le module `react-router-dom` permet de mettre en place un système de routage entre composants.

Installation:

```
npm install react-router-dom@6
```

Routage

```
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import App from "./App";

const root = ReactDOM.createRoot(
  document.getElementById("root")
);
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

Routage

A l’intérieur de <BrowserRouter> nous pouvons définir les routes:

```
<BrowserRouter>
  <Routes>
    <Route path="/" element={<App />}>
    <Route path='/c3/:id' element={<Composant3 />} />
    …..
  </Routes>
</BrowserRouter>
```

Routage

Pour les liens (Link ne recharge pas la page):

```
import { Link } from "react-router-dom";

export class Navigation extends React.Component {
  render() {
    return <nav>
        <Link to='/'>Accueil</Link> |
        <Link to='/c1'>Composant 1</Link> |
        <Link to='/c2'>Composant 2</Link> |
        <Link to='/c3'>Composant 3</Link> |
        <Link to='/c3/45'>Composant 3+</Link>
      </nav>
  }
}
```

Routage

Pour les routes avec paramètres, react-routeur ne fonctionne qu'avec des composants sous forme de fonction

```
import { useParams } from "react-router-dom";

export default function Composant3() {
  let params = useParams();
  console.log(params);
  return <div>
    <p>{params.id}</p>
  </div>
}
```

Routage

Pour les redirections, il faut utiliser useNavigate dans un composant fonction, ou utiliser le component Navigate dans le render d'une classe.

```
render() {
  if(this.state.end === true) {
    return <Navigate to="/c2" />
  }
  return (
    …
  )
}
```