# 10.Créer des formulaires

Les formulaires HTML fonctionnent un peu différemment des autres éléments du DOM en React car ils possèdent naturellement un état interne. Par exemple, ce formulaire en HTML qui accepte juste un nom :

```
<form>
  <label>
    Nom :
    <input type="text" name="name" />
  </label>
  <input type="submit" value="Envoyer" />
</form>
```

Ce formulaire a le comportement classique d'un formulaire HTML et redirige sur une nouvelle page quand l'utilisateur le soumet. Si vous souhaitez ce comportement en React, vous n'avez rien à faire. Cependant, dans la plupart des cas, vous voudrez pouvoir gérer la soumission avec une fonction JavaScript, qui accède aux données saisies par l'utilisateur. La manière classique de faire ça consiste à utiliser les « composants contrôlés ».

## 10.1. Composants contrôlés

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().

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 ».

Par exemple, en reprenant le code ci-dessus pour afficher le nom lors de la soumission, on peut écrire le formulaire sous forme de composant contrôlé :

**Méthode 1 : Composant fonctionnel**

*NameForm.js*

```
import { useState } from "react";

const NameForm = () => {
    const [nom, setNom] = useState('');
    const handleChange =(event)=>{
        setNom(event.target.value);
    }

    const handleSubmit =(event)=>{
        alert('Le nom a été soumis : ' + nom);
        event.preventDefault();
    }
```

```
    return (
    <form onSubmit={handleSubmit}>
        <label>
        Nom :
        <input type="text" name="nom" value={nom}
onChange={handleChange} />
        </label>
        <input type="submit" value="Envoyer" />
    </form>
    );
   };
export default NameForm;
```

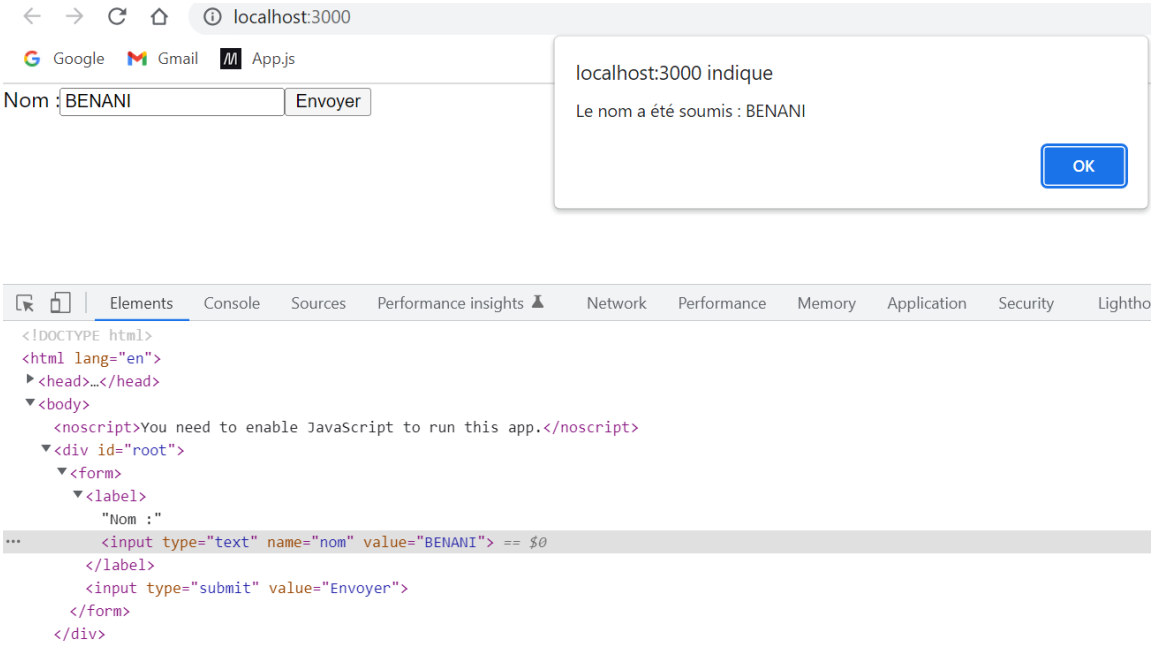
**Méthode 2 : Composant de classe**

*NameForm.js*

```
import React from 'react';

class NameForm extends React.Component {
    constructor(props) {
      super(props);
      this.state = {nom: ''};
      this.handleChange = this.handleChange.bind(this);
      this.handleSubmit = this.handleSubmit.bind(this);
    }
    handleChange(event) {
      this.setState({nom: event.target.value});
    }
    handleSubmit(event) {
      alert('Le nom a été soumis : ' + this.state.nom);
      event.preventDefault();
    }
    render() {
      return (
        <form onSubmit={this.handleSubmit}>
          <label>
            Nom :
            <input type="text" name="nom" value={this.state.nom}
onChange={this.handleChange} />
          </label>
          <input type="submit" value="Envoyer" />
        </form>
      );
    }
}
export default NameForm;
```

Après avoir affiché le composant et tapé quelques caractères dans le champ de saisie, nous obtenons l'affichage suivant :



## 10.2. Gérer les champs de saisie multilignes

Un champ de saisie multiligne se définit grâce à l'élément <textarea>. Créons un composant <TextArea> qui «améliore» l'élément <textarea> traditionnel.

Commençons par un composant de base permettant de créer un élément <textarea> classique. Il s'utilisera sous la forme suivante:

```
<TextArea cols={40} rows={10} value="Tapez votre texte ici" />
```

Comme on l'avait vu précédemment pour les champs de saisie, il faut implémenter l'événement onChange afin que la saisie soit prise en compte dans le champ. De plus, le state doit comporter la valeur saisie dans le champ, afin que cette valeur soit automatiquement rafraîchie lors de la saisie.

La propriété value définie dans le composant devra donc servir à initialiser la propriété value du state.

**Implémentation du composant <TextArea> de base**

**Méthode 1 : Composant fonctionnel**

*TextArea.js*

```
import { useState } from "react";
const FTextArea = (props) => {
    const [message, setMessage] = useState(props.value);
    const handlerChange=(event)=>{
        setMessage(event.target.value);
    }
    const handlerFocus=()=>{
        setMessage("");
    }
    return (
        <textarea cols={props.cols}
        rows={props.rows}
        value={message}
        onFocus={handlerFocus.bind(this)}
        onChange={handlerChange.bind(this)} />
    );
   };
export default FTextArea;
```

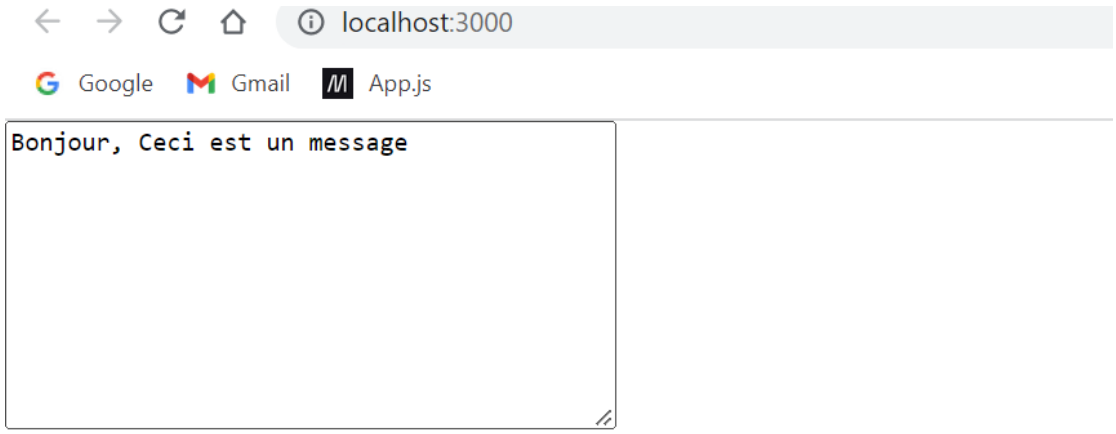
**Méthode 2 : Composant de classe**

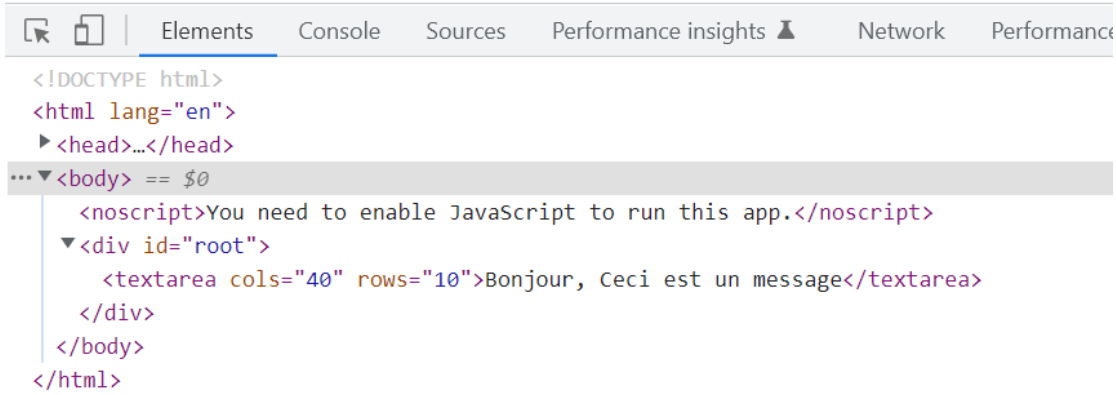
*TextArea.js*

```
import React from 'react';
class TextArea extends React.Component {
    constructor(props) {
        super(props);
        this.state = { message : props.value };
    }
    handlerChange(event) {
```

```
        this.setState({message : event.target.value});
    }
    handlerFocus(event) {
        this.setState({message : ""});
    }
    render() {
        return (
            <textarea cols={this.props.cols}
            rows={this.props.rows}
            value={this.state.message}
            onFocus={this.handlerFocus.bind(this)}
            onChange={this.handlerChange.bind(this)} />
        )
    }
}
export default TextArea;
```

Après avoir affiché le composant et tapé quelques caractères dans le champ de saisie, nous obtenons l'affichage suivant :



## 10.3. Gérer les listes de sélection

Les listes de sélection correspondent aux éléments <select> intégrant les éléments <option> décrivant les items de la liste. Par exemple, voici une liste écrite en HTML intégrant cinq éléments de liste :

```
<select>
  <option value="1">Element1</option>
  <option value="2">Element2</option>
  <option value="3">Element3</option>
  <option value="4">Element4</option>
  <option value="5">Element5</option>
</select>
```

On utilise pour cela un composant <Select> dans lequel on transmet une propriété options qui est un tableau indiquant la liste des éléments à afficher (sous forme de chaînes de caractères). Les éléments React sont créés avec JSX.

**Méthode 1 : Composant fonctionnel**

*Select.js*

```
const Select = (props) => {
    const handlerChange=(event)=>{
        console.log('key='+event.target.value);
    }
    return (
        <select onChange={handlerChange.bind(this)}>
            {props.options.map(function(option, index) {
                return <option key={index+1}
value={index+1}>{option}</option>
            })
            }
        </select>
    );
};
export default Select;
```

**Méthode 2 : Composant de classe**

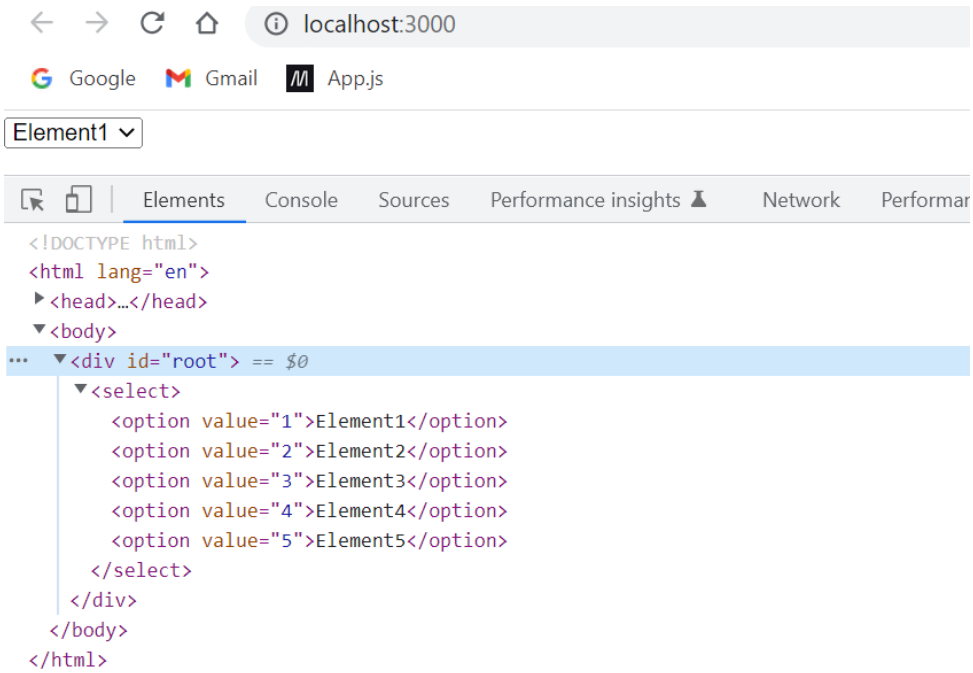
*Select.js*

```
import React from 'react';
class Select extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return (
            <select>
                {this.props.options.map(function(option, index) {
                    return <option key={index+1}
value={index+1}>{option}</option>
                })
                }
            </select>
        )
    }
}
export default Select;
```

Après avoir affiché le composant

```
<Select options={["Element1", "Element2", "Element3", "Element4",
"Element5"]} />
```

Nous obtenons l'affichage suivant :



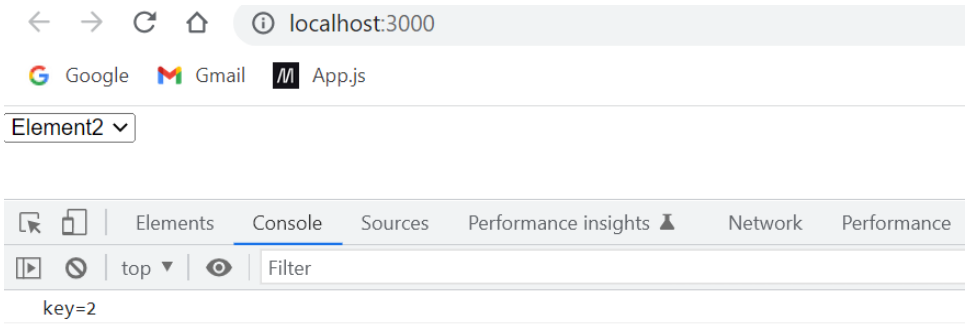
**Afficher l'attribut value de l'élément sélectionné**

On utilise l'attribut onChange sur l'élément <select> afin d'effectuer le traitement lors de la sélection d'un nouvel élément dans la liste.

```
import React from 'react';
class Select extends React.Component {
    constructor(props) {
        super(props);
    }
    handlerChange(event) {
        console.log('key='+event.target.value);
    }
    render() {
        return (
            <select onChange={this.handlerChange.bind(this)}>
                {this.props.options.map(function(option, index) {
                    return <option key={index+1}
value={index+1}>{option}</option>
                })
                }
            </select>
        )
    }
}
export default Select;
```

La valeur de l'élément sélectionné est récupérée au moyen de ***event.target.value***

Après avoir sélectionné plusieurs fois des éléments dans la liste, on obtient l'affichage suivant :




Les valeurs associées aux éléments sélectionnés sont affichées dans la console.

Sélectionner l'élément de liste dont l'attribut value vaut 4

```
<Select defaultValue={4} />
```

## 10.4. Gérer les boutons radio

Nous utilisons ici un seul composant React nommé <RadioGroup> qui est chargé de gérer les boutons radio qui lui sont transmis. Le composant est utilisé de la façon suivante:

**Méthode 1 : Composant fonctionnel**

*RadioGroup.js*

```
const FRadioGroup = (props) => {
    return (
        <div> {
            props.radios.map((radio, index) => {
                return (
                    <label key={index+1}><span>{radio.text}</span>
                    <input type="radio" value={radio.value}
name="radioname" />
                    </label>
                )
            }) }
        </div>
    );
};
export default FRadioGroup;
```

**Méthode 2 : Composant de classe**

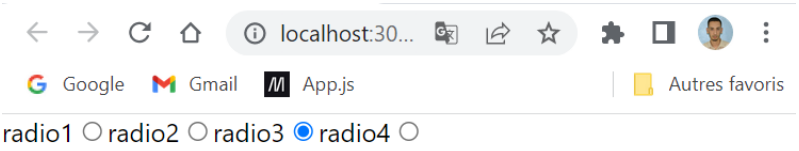
*RadioGroup.js*

```
import React from 'react';
class RadioGroup extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return (
            <div> {
                    this.props.radios.map((radio, index) => {
                        return (
                            <label
key={index}><span>{radio.text}</span>
                            <input type="radio" value={radio.value}
name={radio.name} checked={radio.checked} />
                            </label>
                        )
                    }) }
            </div>
        )
    }
}
export default RadioGroup;
```

Après avoir affiché le composant

```
import RadioGroup from './RadioGroup';
var radios = [
  { value : 1, text : "radio1" },
  { value : 2, text : "radio2" },
  { value : 3, text : "radio3", checked : true },
  { value : 4, text : "radio4" }
];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<RadioGroup radios={radios} name="group1" />);
```

On obtient l'affichage suivant :




Chaque bouton sélectionné désélectionne le bouton précédemment sélectionné.

## 10.5. Gérer les cases à cocher

La gestion des cases à cocher est similaire à celle des boutons radio. Cependant, plusieurs cases à cocher peuvent être sélectionnées en même temps, contrairement aux boutons radio d'un même groupe.

Nous utiliserons pour cela deux composants React:

- le composant <CheckBox> permettra de gérer une case à cocher (via this.state.checked) ;
- le composant <CheckBoxGroup> permettra de gérer l'ensemble des cases à cocher à afficher.

*CheckBoxGroup.js*

```
import React from "react";
class CheckBox extends React.Component {
    constructor(props) {
        super(props);
        this.state = { checked : props.checked || false };
    }
    handlerChange(event) {
        this.setState({checked : event.target.checked});
    }
    render() {
        return (
            <label>
                <span>{this.props.text}</span>
                    <input type="checkbox" value={this.props.value}
                    checked={this.state.checked}
                    onChange={this.handlerChange.bind(this)} />
                <br/>
            </label>
        )
    }
}
class CheckBoxGroup extends React.Component {
    constructor(props) {
        super(props);
    }
    render() {
        return (
            <div>{
                this.props.checkboxes.map((checkbox, index) => {
                    return (
                        <CheckBox key={index} text={checkbox.text}
value={checkbox.value}
                        checked={checkbox.checked}
                        />
                    )
```

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

Après avoir affiché le composant

```
import CheckBoxGroup from './CheckBoxGroup';
var checkboxes = [
  { value : 1, text : "check1" },
  { value : 2, text : "check2", checked : true },
  { value : 3, text : "check3", checked : true },
  { value : 4, text : "check4" }
  ];
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<CheckBoxGroup checkboxes={checkboxes} />);
```

On obtient l'affichage suivant :