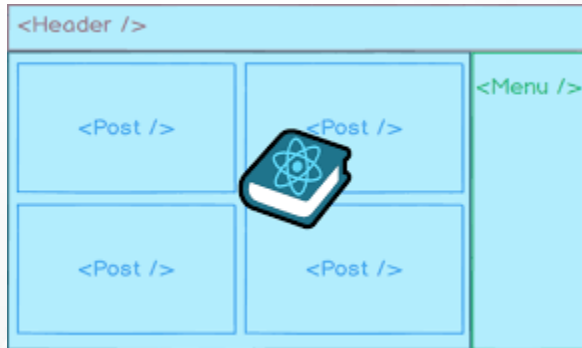


# Chapitre 3 : Les states



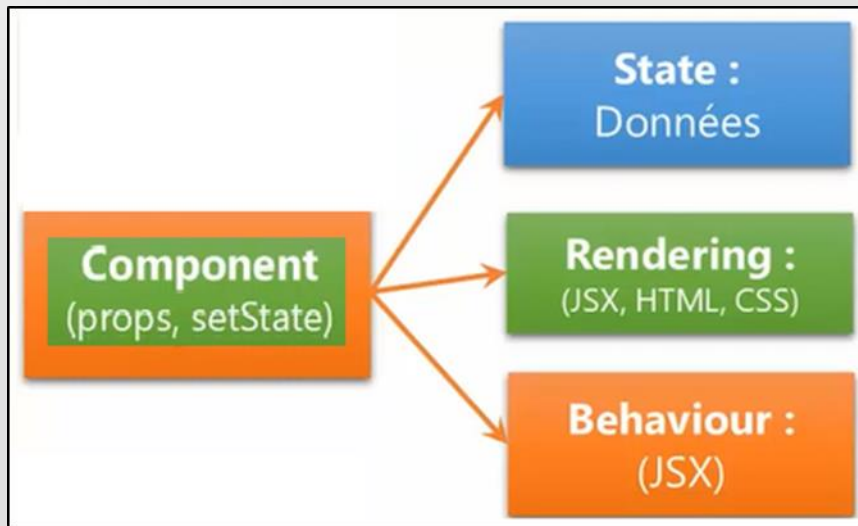
# Le composant

- Le concept le plus important à comprendre en React est le composant.
- En effet, React est conçu autour du concept de composants réutilisables.
- On définit de petits composants et on les met ensemble pour former de plus gros composants.
- Tous les composants, petits ou grands, sont réutilisables, même pour différents projets.
- Un composant React peut être représenté par :
  - Une **fonction** JavaScript : **Statless Component**
  - Une **classe** héritant de la classe React.Component : **Statful Component**
- Un composant React est défini par :
  - Son état (State et Props)
  - Son Rendu (JSX, Html, CSS)
  - Son comportement (JSX)

# Structure d'un Composant React

► Un composant React est défini par :

- Son état (State et Props)
- Son Rendu (JSX, HTML, CSS)
- Son comportement (JSX)



# Composant fonctionnel 1/2

- ▶ Si on écrit un composant React qui ne nécessite pas d'état et qu'on souhaite créer une interface utilisateur réutilisable, on peut l'écrire en tant que composant fonctionnel sans état.

## Exemple :

- *Fichier App.js*

```
import React from 'react';
import ReactDOM from 'react-dom';
function App() {
  return <h1>Client : Sotratex</h1>;
}
const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);
export default App;
```

# Composant fonctionnel 2/2

- *index.html*

```
<body>  
  <div id="root"></div>  
</body>
```

Ce qui donne le résultat suivant :

**Client : Sotratex**

# Composant à base de classe 1/3

- ▶ On peut utiliser une classe ES6 pour définir un composant.
- ▶ Le nom du composant doit commencer par une lettre majuscule.
- ▶ Le composant doit inclure l'instruction extend React.Component, cette instruction crée un héritage à React.Component et donne au composant l'accès aux fonctions de React.Component.
- ▶ Le composant nécessite également une méthode render(), cette méthode retourne du HTML.

# Composant à base de classe 2/3

## Exemple :

- *Fichier App.js*

```
import React from 'react';
import ReactDOM from "react-dom";
class App extends React.Component {
  render() {
    return (
      <h1>Bonjour</h1>
    );
  }
}

const rootElement = document.getElementById("root");
ReactDOM.render(<App />, rootElement);

export default App;
```

# Composant à base de classe 3/3

- *index.html*

```
<body>  
  <div id="root"></div>  
</body>
```

Ce qui donne le résultat suivant : Bonjour



# Éléments d'un composant React 1/3

- La méthode `render()` ne retourne qu'un seul élément, dans cet exemple, on veut afficher plusieurs composants HTML.

```
class App extends React.Component {  
  render() {  
    return (  
      <h1>Bonjour</h1>  
      <ul>Client1</ul>  
    );  
  }  
}
```



- On voit que ça génère une erreur, cet exemple permet de retourner deux éléments HTML (un élément `h1` et un élément `ul`) ce qui n'est pas possible puisque la fonction `render()` ne retourne qu'un seul élément.

# Éléments d'un composant React 2/3

## ► Solution :

Placer les éléments dans une seule division (<div>) :

```
class App extends React.Component {  
  render() {  
    return (  
      <div className="App">  
        <h1>Liste des clients</h1>  
        <ul> <li>Sotratex<button>X</button></li>  
          <li>Sios Zitex<button>X</button></li>  
          <li>Elmazraa<button>X</button></li> </ul>  
        <form> <input type="text" placeholder="Ajouter un client"/>  
          <button>Confirmer</button>  
        </form>  
      </div>  
    );  
  }  
}
```

# Éléments d'un composant React 3/3

- ▶ Un composant App est un ensemble d'éléments React (h1, ul, li, etc.).

## Liste des clients

- Sotratex ☐
- Sios Zitex ☐
- Elmazraa ☐

# Interpolation JSX 1/3

- ▶ Déclarer une variable et l'afficher :

```
class App extends React.Component {  
  render() {  
    const title="liste des clients"  
    return (  
      <div>  
        <h1> {title} </h1>  
      </div>  
    );  
  }  
}
```

**liste des clients**

- ▶ Ce qui donne le résultat suivant :

# Interpolation JSX 2/3

► Une variable peut contenir un élément JSX et être utilisée dans le rendu :

```
class App extends React.Component {  
  render() {  
    const title="liste des clients";  
    const element = <li>Sotratex<button>X</button></li>  
    return (  
      <div>  
        <h1> {title} </h1>  
        <ul> {element} </ul>  
  
        <form> <input type="text" placeholder="Ajouter un client" />  
          <button>Confirmer</button> </form>  
      </div>  
    );  } }
```

## liste des clients

- Sotratex X

ce qui donne le résultat suivant

# Interpolation JSX 3/3

► Une variable peut contenir un tableau d'élément JSX et être utilisée dans le rendu.

```
class App extends React.Component {  
  render() {  
    const title="liste des clients";  
    const tabelemnts=[  
      <li>Janvier</li>,  
      <li>Fevrier</li>,  
      <li>Mars</li>    ]  
  
    return (  
      <div>  
        <h1> {title} </h1>  
        <ul> {tabelemnts} </ul>  
        <form> <input type="text" placeholder="Ajouter un client" />  
          <button>Confirmer</button> </form>  
      </div>  
    );  }  }
```

## liste des clients

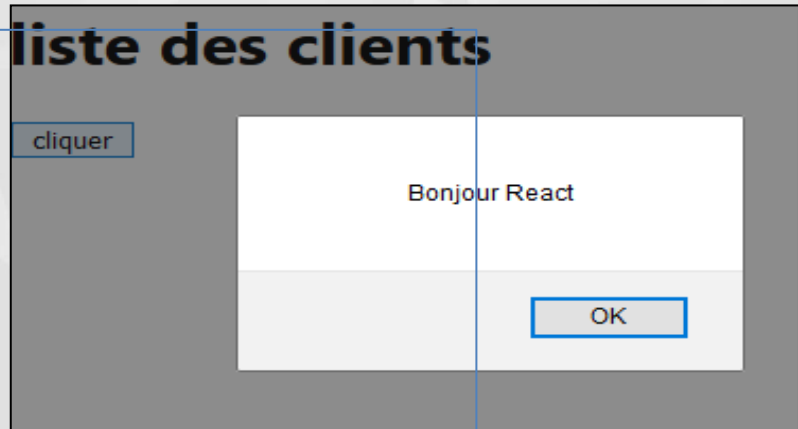
- Janvier
- Fevrier
- Mars

Ce qui génère ce résultat dans le navigateur

# Événements dans ReactJs

- ▶ Créer une fonction qui va faire un alert :

```
class App extends React.Component {  
  handleClick(){  
    alert("Bonjour React")  
  }  
  render() {  
    const title="liste des clients";  
    return (  
      <div>  
        <h1> {title} </h1>  
        <button onClick={this.handleClick}>cliquer </button>  
      </div>  
    );  
  }  
}
```



- ▶ Donnant le résultat suivant en cliquant sur le bouton « cliquer » :

# Appel de composants dans d'autres 1/4

- React consiste à réutiliser du code, et il peut être judicieux d'insérer certains des composants dans des fichiers séparés.
- Les composants peuvent donc faire référence à d'autres composants dans leur sortie.
- Ça nous permet d'utiliser la même abstraction de composants pour n'importe quel niveau de détail.
- Pour ce faire, on va créer deux nouveaux fichiers avec une extension de fichier .js appelés respectivement First et Second.
- Notez que les fichiers doivent commencer par importer React et qu'ils doivent se terminer par l'instruction export default.



# Appel de composants dans d'autres 2/4

- *Composant First.js*

```
import React from 'react';
class First extends React.Component {
  render() {
    return (
      <div>
        <h1>Contenu du composant First</h1>
      </div>
    );
  }
}
export default First;
```

# Appel de composants dans d'autres 3/4

- *Composant Second.js*

```
import React from 'react';  
class Second extends React.Component {  
  render() {  
    return (  
      <div>  
        <h1>Contenu du composant Second</h1>  
      </div>  
    );  
  }  
}  
export default Second;
```

# Appel de composants dans d'autres 4/4

► Pour pouvoir utiliser les composants First et Second, on doit importer les fichiers correspondants dans l'application.

- *Composant App.js*

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

**Contenu du composant First**

**Contenu du composant Second**

Le résultat obtenu est le suivant

# Création de l'objet State 1/4

- Les composants React ont un objet d'état intégré (state).
- L'objet d'état est l'endroit où on stocke les valeurs de propriété qui appartiennent au composant.
- Lorsque l'objet d'état change, le composant effectue un nouveau rendu.
- L'objet state est initialisé dans le constructeur.
- La méthode `map` permet de créer un tableau contenant des éléments transformés d'un autre tableau.
- La fonction qui va boucler sur le tableau client et qui va appeler la fonction indiquée.

# Création de l'objet State 2/4

```
class App extends React.Component {  
  state = { clients: [  
    { id: 1, nom: "Sotratex" },  
    { id: 2, nom: "Sios-Zitex" },  
    { id: 3, nom: "Elmazraa" }  ] };  
  
  render() {  
    const title="liste des clients";  
    return ( <div>  
      <h1>{title}</h1>  
      <ul> {this.state.clients.map(client=>(  
        <li>{client.nom}<button>X</button></li> ))} </ul>  
      <form> <input type="text" placeholder="Ajouter un client" />  
        <button>Confirmer</button> </form>  
    </div>  
  ); } }
```

## liste des clients

- Sotratex X
- Sios-Zitex X
- Elmazraa X

# Création de l'objet State 3/4

## ► Remarque :

On peut utiliser la fonction fléchée dans la déclaration du tableau d'éléments.

```
const elements = this.state.clients.map((client)=>  
  <li>{client.nom}<button>X</button></li>  )
```

## Exemple :

```
state = {  
  clients: [  
    { id: 1, nom: "Sotratex" },  
    { id: 2, nom: "Sios-Zitex" },  
    { id: 3, nom: "Elmazraa" }  
  ] };  
handleClick(){  
  console.log(this.state);  
}
```

► Si on appui sur le bouton, rien ne s'affiche parce que `this` ne représente pas le composant.

► Quand on lie une fonction à un événement, `this` à l'intérieur de la fonction ne représente pas l'objet global, le `this` représente l'élément auquel on a déclenché l'événement.

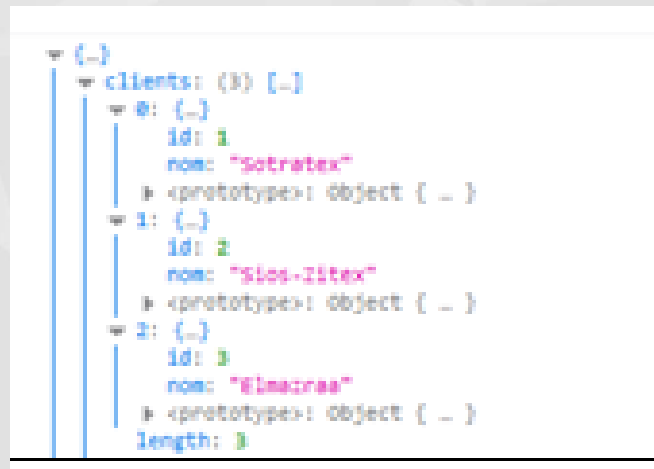
# Création de l'objet State 4/4

► Solution : Utilisation de la fonction fléchée : Ce qui donne le résultat suivant dans la console :

```
class App extends React.Component {  
  state = { clients: [  
    { id: 1, nom: "Sotratex" },  
    { id: 2, nom: "Sios-Zitex" },  
    { id: 3, nom: "Elmazraa" }  
  ] };  
}
```

```
handleClick={()=>{  
  console.log(this.state);  
}}
```

```
render() { const title="liste des clients";  
  return ( <div>  
    <h1>{title}</h1>  
    <button onClick={this.handleClick}>cliquer </button>  
  </div>  
); }
```



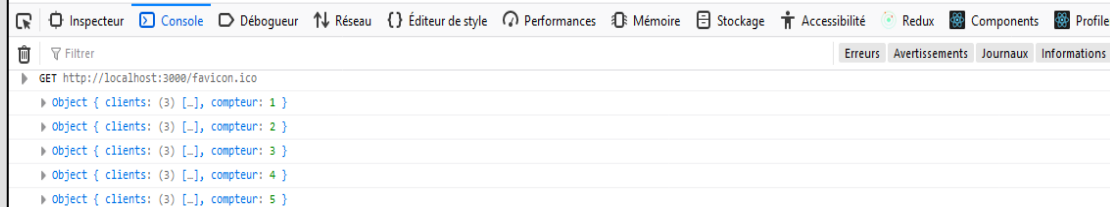
# Modifier un state 1/6

- Lorsqu'une valeur de l'objet state change, le composant est rendu de nouveau, ce qui signifie que la sortie change en fonction de la ou des nouvelles valeurs.
- On va écrire un script react qui va initialiser un compteur, et lorsqu'on appuie sur le bouton le compteur sera incrémenté,
- Voilà le résultat qu'on devrait obtenir :

## liste des clients

0

cliquer





# Modifier un state 2/6

```
class App extends React.Component {  
  state = { clients: [  
    { id: 1, nom: "Sotratex" },  
    { id: 2, nom: "Sios-Zitex" },  
    { id: 3, nom: "Elmazraa" } ],  
    compteur: 0  };  
  handleClick = () => {  
    this.state.compteur++;  
    console.log(this.state);  
  };  
  render() {  
    const title = "liste des clients";  
    return ( <div>  
      <h1>{title}</h1> <h3> {this.state.compteur} </h3>  
      <button onClick={this.handleClick}>cliquer </button>  
    </div>  
  );  } }
```

# Modifier un state 3/6

- Dans cet exemple la valeur compteur va changer dans la console mais ne va pas être changée dans le navigateur, la valeur 0 reste inchangée.
- React est incapable de suivre les changements des valeurs state.
- React ne surveille pas les variables, il ne les voit pas changer.
- Il faut donc faire appel à une méthode qui non seulement va modifier le state, mais qui va encore prévenir React pour réafficher le rendu.
- Cette méthode s'appelle **setState()**

# Modifier un state 4/6

setState() permet donc de modifier l'état du composant et d'en informer react.

```
handleClick = () => {  
  this.setState({compteur : this.state.compteur + 1})  
  console.log(this.state);  
};
```

► Le résultat obtenu est :



# Modifier un state 5/6


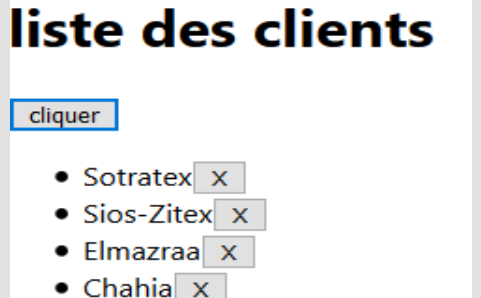
## Exemple :

On veut écrire un script react qui permet d'ajouter un élément dans le tableau et de l'afficher.  
La méthode slice() permet de créer une copie du tableau clients et le tableau state reste inchangé.

```
class App extends React.Component {
  state = {
    clients: [
      { id: 1, nom: "Sotratex" },
      { id: 2, nom: "Sios-Zitex" },
      { id: 3, nom: "Elmazraa" }
    ]
  };
  handleClick = () => {
    const clients = this.state.clients.slice();
    clients.push({ id: 4, nom: "Chahia" });
    this.setState({ clients: clients });
  };
}
```

# Modifier un state 6/6

```
render() {  
  const title = "liste des clients";  
  return (  
    <div>  
      <h1>{title}</h1>  
      <button onClick={this.handleClick}>cliquer </button>  
      <ul>  
        {this.state.clients.map(client=>(  
          <li>{client.nom}<button>X</button></li>  
        ))}  
      </ul>  
    </div>  
  );  
}
```

Etat 1	Etat 2
	

► Le résultat obtenu est :

# Supprimer un élément 1/3

- ▶ La méthode `findIndex()` permet de trouver la position d'un élément particulier dans un tableau.

```
class App extends React.Component {
  state = {
    clients: [
      { id: 1, nom: "Sotratex" },
      { id: 2, nom: "Sios-Zitex" },
      { id: 3, nom: "Elmazraa" }
    ]
  };
  handleDelete = id => {
    const clients = this.state.clients.slice();
    const index = clients.findIndex(client=>client.id === id);
    clients.splice(index, 1);
    this.setState({ clients: clients });
  };
}
```

# Supprimer un élément 2/3

```
render() {  
  const title = "liste des clients";  
  
  return (  
    <div>  
      <h1>{title}</h1>  
      <ul>  
        {this.state.clients.map(client => (  
          <li>  
            {client.nom}  
            <button onClick={() => this.handleDelete(client.id)}>X</button>  
          </li>  
        ))}  
      </ul>  
    </div>  
  );  
}
```

# Supprimer un élément 3/3

► Le résultat obtenu est le suivant :

Etat 1	Etat 2
<div data-bbox="195 426 821 801"><h2>liste des clients</h2><ul style="list-style-type: none"><li>• Sotratex <input checked="" type="checkbox"/></li><li>• Sios-Zitex <input checked="" type="checkbox"/></li><li>• Elmazraa <input checked="" type="checkbox"/></li></ul></div>	<div data-bbox="983 426 1617 762"><h2>liste des clients</h2><ul style="list-style-type: none"><li>• Sotratex <input checked="" type="checkbox"/></li><li>• Sios-Zitex <input checked="" type="checkbox"/></li></ul></div>