

Technos Web

# REACT



# REACT

## partie 1



# REACT

- > Bibliothèque JavaScript développée par Facebook
- > Depuis 2013
- > Open Source sous licence MIT
- > Très populaire depuis quelques années
- > Utilisable en combinaison avec les frameworks comme Angular
  
- > Objectif :
  - ▶ Faciliter la conception de SPA (Single Page Application)
  - ▶ Création / utilisation de **composants**
  - ▶ Se limite à la création de l'interface HTML
  - ▶ Utilise un DOM virtuel et ne met à jour le rendu dans le navigateur que lorsque c'est nécessaire

- > Evolution constante et très rapide
- > Beaucoup de changements majeurs suivant les versions

Beaucoup de documentation  
obsolète !!!

# CRÉER UN PROJET REACT

- > Exécuter la ligne de commande suivante :

```
1 npx create-react-app react-demo
```

- > Note : **npx** est une commande de npm pour la création de projets
- > Cela crée le dossier **react-demo**
- > Dans le dossier du projet, exécuter la ligne de commande suivante :

```
1 yarn start
```

- > Yarn est un autre gestionnaire de package comme npm.
- > Démarrer un serveur web sur le port 3000 par défaut (développement)
- > Ouvre un navigateur sur l'application de démonstration (logo React qui tourne)
- > Surveillance des modifications de fichiers pour recompilation automatique

# CRÉER UN PROJET REACT



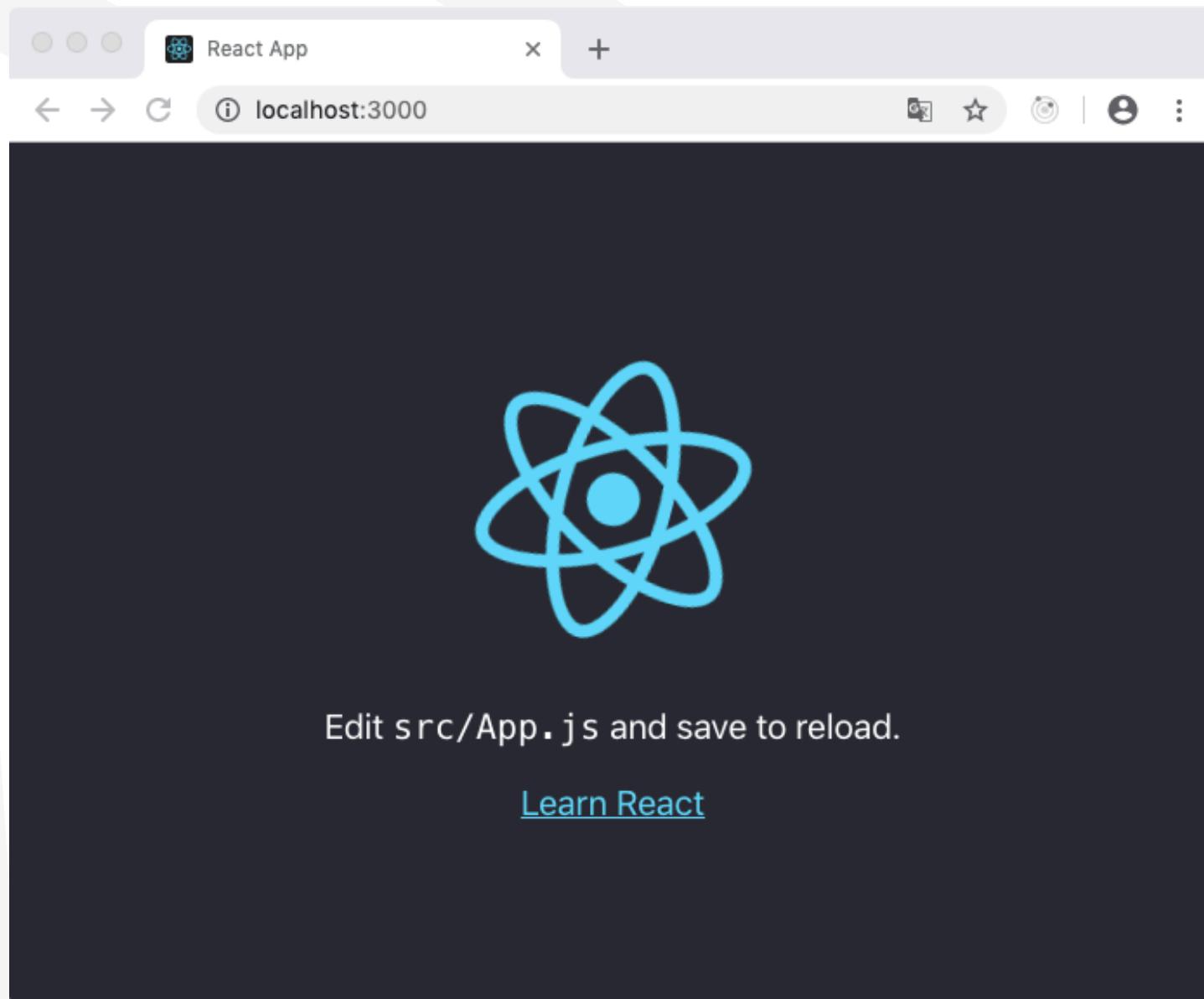
Liste des différentes commandes disponibles avec yarn :

```
1 Success! Created react-demo at
   /Volumes/Data/3iL/www/I3_TechnosWeb/2019-2020-prepa/react-demo
2 Inside that directory, you can run several commands:
3
4   yarn start
5     Starts the development server.
6
7   yarn build
8     Bundles the app into static files for production.
9
10  yarn test
11    Starts the test runner.
12
13  yarn eject
14    Removes this tool and copies build dependencies, configuration files
15    and scripts into the app directory. If you do this, you can't go back!
16
17 We suggest that you begin by typing:
18
19   cd react-demo
20   yarn start
21
22 Happy hacking!
```

# CRÉER UN PROJET REACT



Résultat à obtenir dans un navigateur :



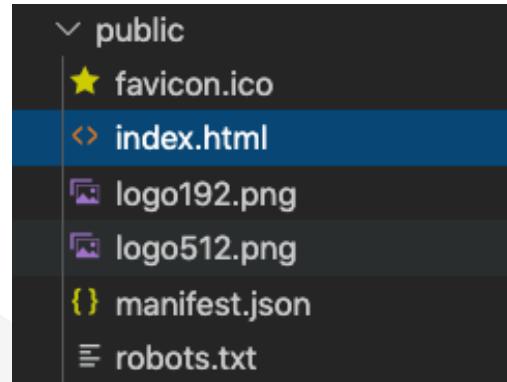
# CRÉER UN PROJET REACT

- > Le code est compilé automatiquement avec Babel et Webpack
- > Webpack permet de rassembler tous les fichiers en un fichier unique.
- > Babel permet de transpiler un code ES 2015+ en un code compatible avec les anciennes version de JS et donc exécutable sur les navigateurs même les plus anciens.

# TOUR DU PROJET

> Le projet contient deux dossiers :

- ▶ /public : base du contenu qui sera publié
- ▶ /src : pour contenir le code source



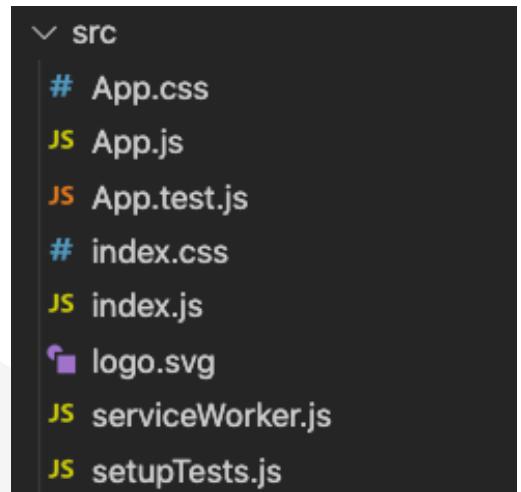
> Dans le fichier `index.html` il y a une balise `<div id="root"></div>` dans laquelle sera rendue l'application React (ligne 31).

/public/index.html

```
29  <body>
30    <noscript>You need to enable JavaScript to run this app.</noscript>
31    <div id="root"></div>
32    <!--
33      This HTML file is a template.
34      If you open it directly in the browser, you will see an empty page.
35
36      You can add webfonts, meta tags, or analytics to this file.
37      The build step will place the bundled scripts into the <body> tag.
38
39      To begin the development, run `npm start` or `yarn start`.
40      To create a production bundle, use `npm run build` or `yarn build`.
41    -->
42  </body>
43 </html>
```

# TOUR DU PROJET

- > Dans le dossier `/src`, le fichier principal est `index.js`
- > C'est lui qui assure la liaison entre le composant React `<App/>` défini dans `App.js` et la balise `<div id="root"></div>`
- > Ce n'est qu'une façon de démarrer un projet React.



`/src/index.js`

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import './index.css';
4 import App from './App';
5 import * as serviceWorker from './serviceWorker';

6

7 ReactDOM.render(<App />, document.getElementById('root'));

8

9 // If you want your app to work offline and load faster, you can change
10 // unregister() to register() below. Note this comes with some pitfalls.
11 // Learn more about service workers: https://bit.ly/CRA-PWA
12 serviceWorker.unregister();
```

# TOUR DU PROJET

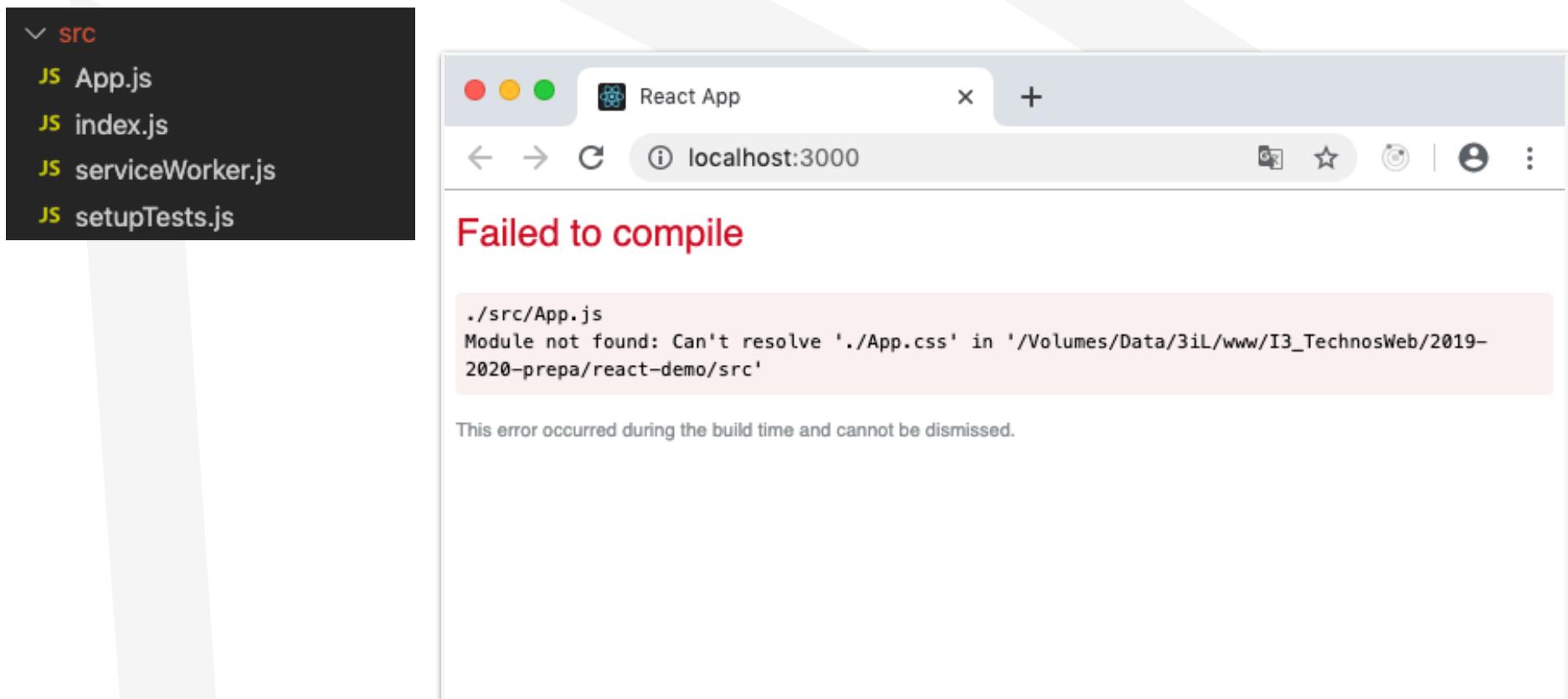
## > Contenu du fichier /src/App.js

/src/App.js

```
1 import React from 'react';
2 import logo from './logo.svg';
3 import './App.css';
4
5 function App() {
6   return (
7     <div className="App">
8       <header className="App-header">
9         <img src={logo} className="App-logo" alt="logo" />
10        <p>
11          | Edit <code>src/App.js</code> and save to reload.
12        </p>
13        <a
14          | className="App-link"
15          | href="https://reactjs.org"
16          | target="_blank"
17          | rel="noopener noreferrer"
18          |
19          | Learn React
20        </a>
21      </header>
22    </div>
23  );
24}
25
26 export default App;
```

# NETTOYAGE DU PROJET

- > Avant d'aller plus loin, nous allons supprimer un certain nombre de fichiers qui ne nous serviront pas.
- > Ne garder que les fichiers ci-dessous dans le dossier **/src**.
- > Dans le navigateur les effets doivent se ressentir.

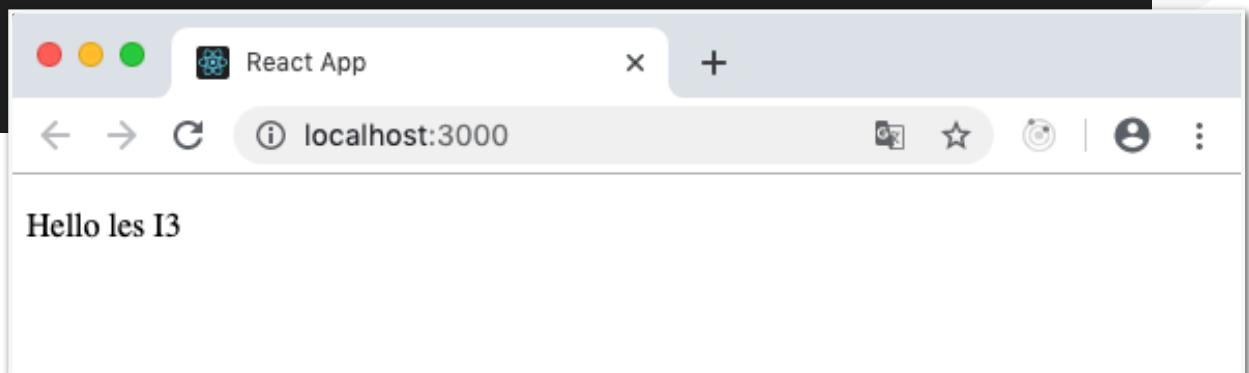


# HELLO WORLD

- > Dans `/src/index.js`, supprimer la ligne d'import de `./index.css`
- > Modifier le fichier `/src/App.js` comme suit :

`/src/App.js`

```
1 import React from 'react';
2
3 function App() {
4   let someone = "les I3";
5
6   return (
7     <div className="App">
8       <p>Hello {someone}</p>
9     </div>
10  );
11}
12
13 export default App;
```



# EXPLICATION DU CODE

/src/App.js

```
1 import React from 'react';
2
3 function App() {
4   let someone = "les I3";
5
6   return (
7     <div className="App">
8       <p>Hello {someone}</p>
9     </div>
10  );
11}
12
13 export default App;
```

- > Ligne 1 : importation de la bibliothèque React.
- > Ligne 3 : création du composant App. Ici composant dit fonctionnel.
- > Lignes 7 à 9 : code JSX, propre à React, sert à produire du HTML.
- > Ligne 8 : {someone} intégration de la variable someone dans le code JSX.
- > Ligne 13 : exporte le composant App (pour qu'il soit utilisable par d'autres scripts, notamment index.js). Très important.

- > JSX extension de syntaxe JavaScript
- > Aucune obligation de l'utiliser, mais simplifie l'écriture

JSX n'est pas du HTML  
ni du XML

# REACT - JSX

- > Objectif de React : simplifier la création d'application HTML / JavaScript par le biais de **composants**.
- > Composant : brique HTML/Javascript "réutilisable" (comme `<App/>`)
- > **Important** : les fichiers js utilisés au développement sont compilés.
- > Utilisation du code JSX pour se rapprocher au plus près du HTML avec des possibilités en plus.
  - ▶ Reprend en partie la syntaxe HTML
  - ▶ Modification de certains noms d'attributs comme **class** (HTML) qui devient **className** (JSX) car le mot **class** existe en JavaScript
  - ▶ Idem pour **for** (utilisé dans `<label>`) qui devient **forHtml**.

# EXPLICATION DU CODE

/src/App.js

```
6  return (
7    <div className="App">
8      <p>Hello {someone}</p>
9    </div>
10 );
```

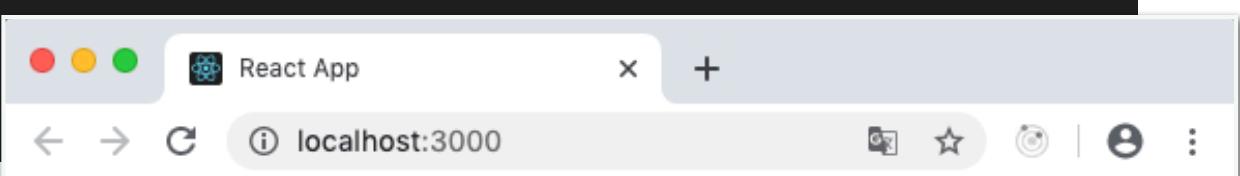
- > Ligne 7 : <div> n'est pas la balise <div> HTML, mais le composant <div> de React, dont le rendu est une balise <div> HTML.
- > Ligne 8 : idem pour <p>.
- > Noter les parenthèses autour du code JSX.

# EXPLICATION DU CODE

- > Règle importante : le code JSX n'admet qu'un seul composant racine.

/src/App.js

```
6  return (
7    <div className="App">
8      <p>Hello {someone}</p>
9    </div>
10   <p>Coucou</p>
11 );
12 }
```



Failed to compile

```
./src/App.js
Line 10:5:  Parsing error: Adjacent JSX elements must be wrapped in an enclosing
tag. Did you want a JSX fragment <>...</>?

8 |       <p>Hello {someone}</p>
9 |
10|   <p>Coucou</p>
> 11|   ^
|
12| );
13| 
```

This error occurred during the build time and cannot be dismissed.

- > Observer le message d'erreur.

# EXPLICATION DU CODE

/src/App.js

```
6  return (
7    <div className="App">
8      <p>Hello {someone}</p>
9    </div>
10 );
```

- > Ligne 8 : les accolades permettent d'insérer des expressions JavaScript au sein du code JSX.
- > Attention à la subtilité "expression JavaScript" VS "code JavaScript".
- > Une expression JavaScript a résultat (qui peut être null ou vide). Typiquement un calcul ou un appel de fonction (ou un calcul contenant un appel de fonction).
- > Les expressions entre accolades sont systématiquement échappées pour éviter les failles de types XSS.

# REACT

## partie 2

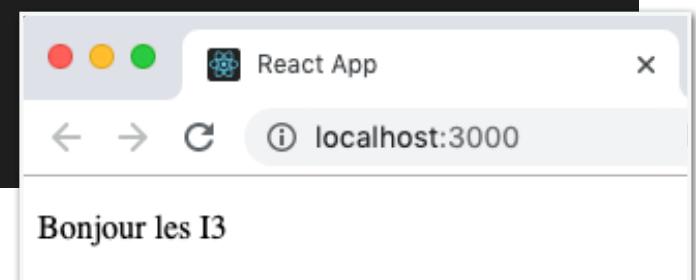


# LES PROPS

- > L'équivalent des attributs HTML s'appelle **props** dans le monde React.
- > Une **props** est considérée comme une entrée pour un composant.
- > Exemple définissant un composant **<Bonjour >**:

/src/App.js

```
1 import React from 'react';
2
3 function Bonjour(props) {
4   return <p>{props.salutation} {props.qui}</p>
5 }
6
7 function App() {
8   let someone = "les I3";
9
10  return (
11    <div className="App">
12      <Bonjour salutation="Bonjour" qui={someone}/>
13    </div>
14  );
15}
16
17 export default App;
```



# LES PROPS

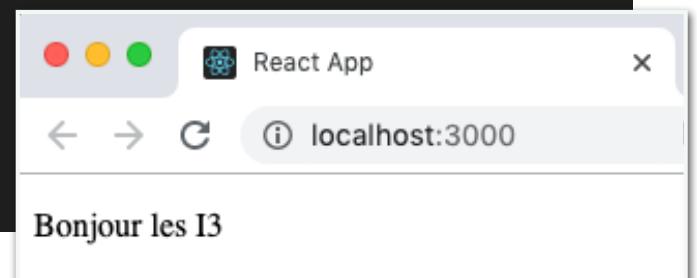
- > Par défaut un composant n'effectue aucun contrôle sur ses props.
- > Dans l'exemple précédent, ni **salutation**, ni **qui** ne sont spécifiés comme attendus par le composant.
  - ▶ Il y a des possibilités pour imposer des props particulières.
  - ▶ Une props attendue par un composant mais non renseignée vaut **undefined**.
- > Les attributs HTML sont repris sous la forme de props pour les composants équivalents (attention aux majuscules).

# COMPOSANT SOUS FORME DE CLASSE

- > En React un composant peut être défini sous forme de fonction ou de classe.
- > Les fonctions sont pour les composants simples, les classes pour des composants plus complexes.

/src/App.js

```
1 import React, { Component } from 'react';
2
3 class Bonjour extends Component
4 {
5   render() {
6     return <p>{this.props.salutation} {this.props.qui}</p>
7   }
8 }
9
10 function App() {
11   let someone = "les I3";
12
13   return (
14     <div className="App">
15       <Bonjour salutation="Bonjour" qui={someone}/>
16     </div>
17   );
18 }
19
20 export default App;
```



- > Attention à la ligne 1 et à l'héritage.

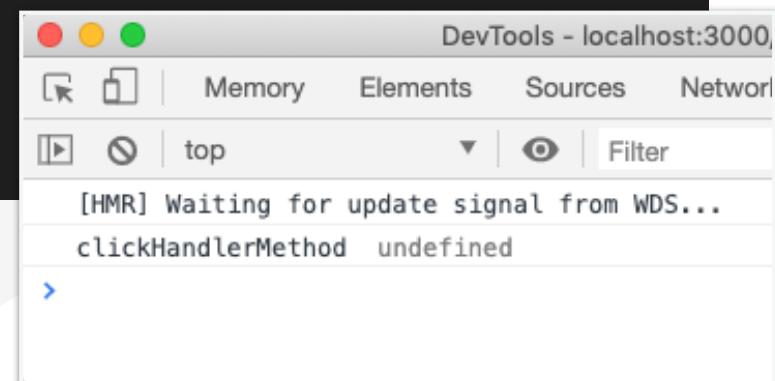
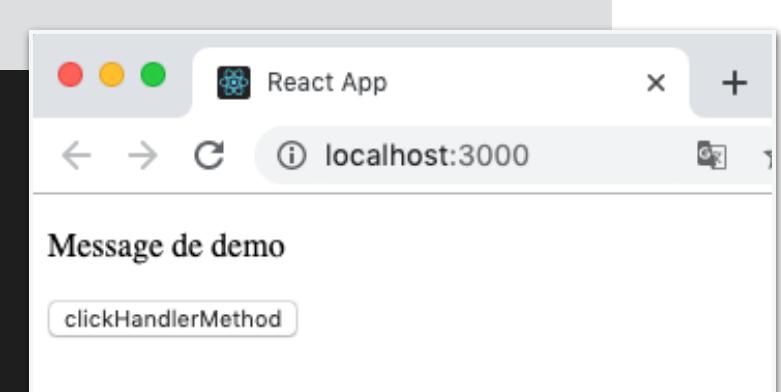
# GÉRER UN ÉVÉNEMENT



Spontanément nous aurions envie d'écrire :

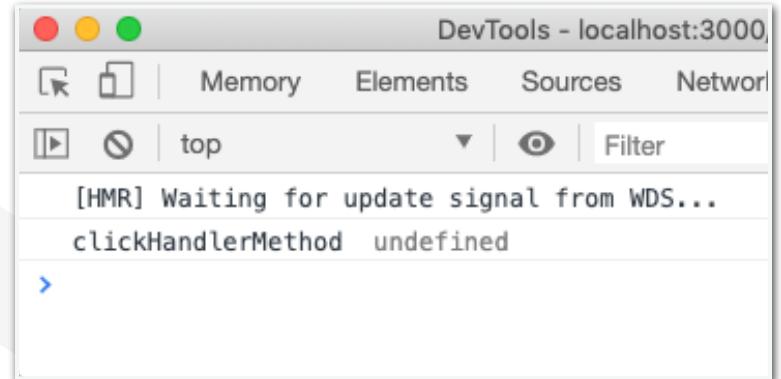
/src/App.js

```
1 import React, { Component } from 'react';
2
3 class App extends Component
4 {
5   constructor(props) {
6     super(props);
7     this.message = "Message de demo"
8   }
9
10  clickHandlerMethod () {
11    console.log("clickHandlerMethod ",this);
12  }
13
14  render() {
15    return (<div>
16      <p>{this.message}</p>
17      <button onClick={this.clickHandlerMethod}>clickHandlerMethod</button>
18    </div>)
19  }
20}
21
22 export default App;
```



# GÉRER UN ÉVÉNEMENT

- > Quel est le souci ?
- > Le gestionnaire d'événement est bien lancé mais `this` vaut `undefined`. Pourquoi ?
- > Cela vient de la traduction faite par React de notre composant.
- > L'écriture `onClick={this.clickHandler}` ne signifie pas l'appel de la méthode `clickHandler()` à partir de l'objet courant, mais utilise "en tant que fonction" la méthode `clickHandler()`, mais sans son contexte.
- > Il faut redonner un contexte d'appel lors de la gestion d'événement. Cela se fait en JavaScript au moyen de `bind()` (vu dans les cours sur JavaScript) ou au travers d'une écriture différente du gestionnaire d'événement.



# GÉRER UN ÉVÉNEMENT

> En gardant "de vraies méthodes" :

/src/App.js

```
1 import React, { Component } from 'react';
2
3 class App extends Component
4 {
5   constructor(props) {
6     super(props);
7     this.message = "Message de demo"
8
9     this.clickHandlerMethod_V2 = this.clickHandlerMethod_V2.bind(this);
10 }
11
12 clickHandlerMethod () {
13   console.log("clickHandlerMethod ",this); }
14
15 clickHandlerMethod_V2 () {
16   console.log("clickHandlerMethod_V2 ",this); }
17
18 render() {
19   return (<div>
20     <p>{this.message}</p>
21     <button onClick={this.clickHandlerMethod}>clickHandlerMethod</button>
22     <button onClick={this.clickHandlerMethod.bind(this)}>clickHandlerMethod avec
23       bind</button>
24     <button onClick={this.clickHandlerMethod_V2}>clickHandlerMethod_V2</button>
25   </div>);
26 }
27 export default App;
```

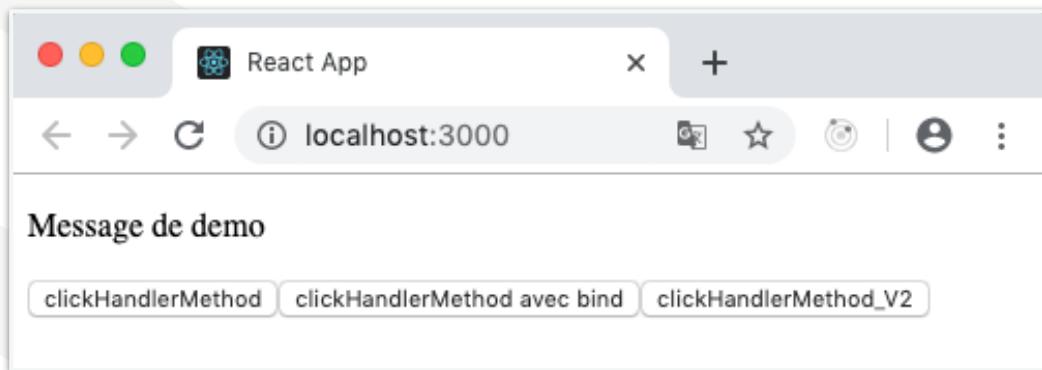
binding effectué "une fois pour toutes" dans le constructeur

binding effectué "à l'usage" (ligne 22)

# GÉRER UN ÉVÉNEMENT



Résultat (après appui sur chacun des boutons)



À retenir : pour qu'une méthode "classique" puisse servir de gestionnaire d'événement, il faut opérer un **bind()** soit à l'usage, soit dans le constructeur (meilleure solution).

# GÉRER UN ÉVÉNEMENT

> En utilisant des fonctions (pouvant être attachées à des propriétés) :

/src/App.js

```
3  class App extends Component
4  {
5    constructor(props) {
6      super(props);
7      this.message = "Message de demo"
8    }
9
10   clickHandlerMethod () {
11     console.log("clickHandlerMethod ",this);
12   }
13
14   clickHandlerFunction = (event) => {
15     console.log("clickHandlerFunction ",this);
16   }
17
18   render() {
19     const clickHandlerInternalFunction = () => {
20       console.log("clickHandlerInternalFunction ",this)
21     }
22
23     return (<div>
24       <p>{this.message}</p>
25       <button onClick={this.clickHandlerMethod}>clickHandlerMethod</button>
26       <button onClick={this.clickHandlerFunction}>clickHandlerFunction</button>
27       <button onClick={clickHandlerInternalFunction}
28         >clickHandlerInternalFunction</button>
29     </div>);
30 }
```

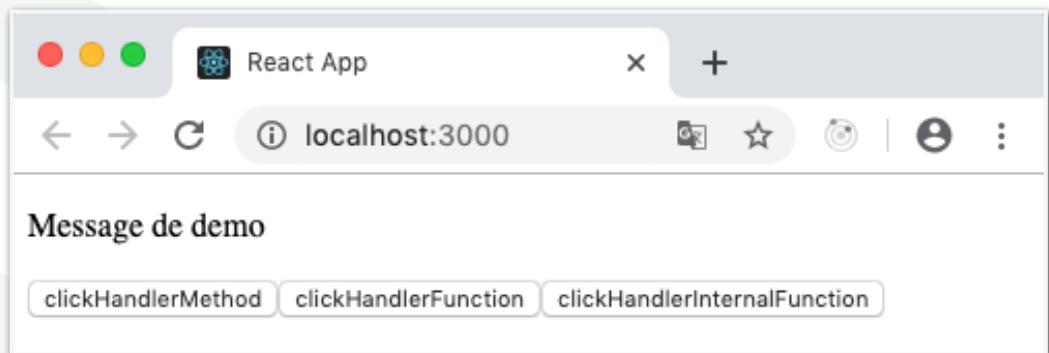
The diagram illustrates three types of event handlers:

- An arrow points from the annotation "Fonction attachée à une propriété" to the line `onClick={this.clickHandlerMethod}`.
- An arrow points from the annotation "Simple fonction" to the line `onClick={this.clickHandlerFunction}`.
- An arrow points from the annotation "Aucun this" to the line `onClick={clickHandlerInternalFunction}`.

# GÉRER UN ÉVÉNEMENT



Résultat (après appui sur chacun des boutons)



Aucune stratégie globale à choisir (méthode classique ou fonction), le contexte peut imposer l'usage de méthodes

# REACT

## partie 3

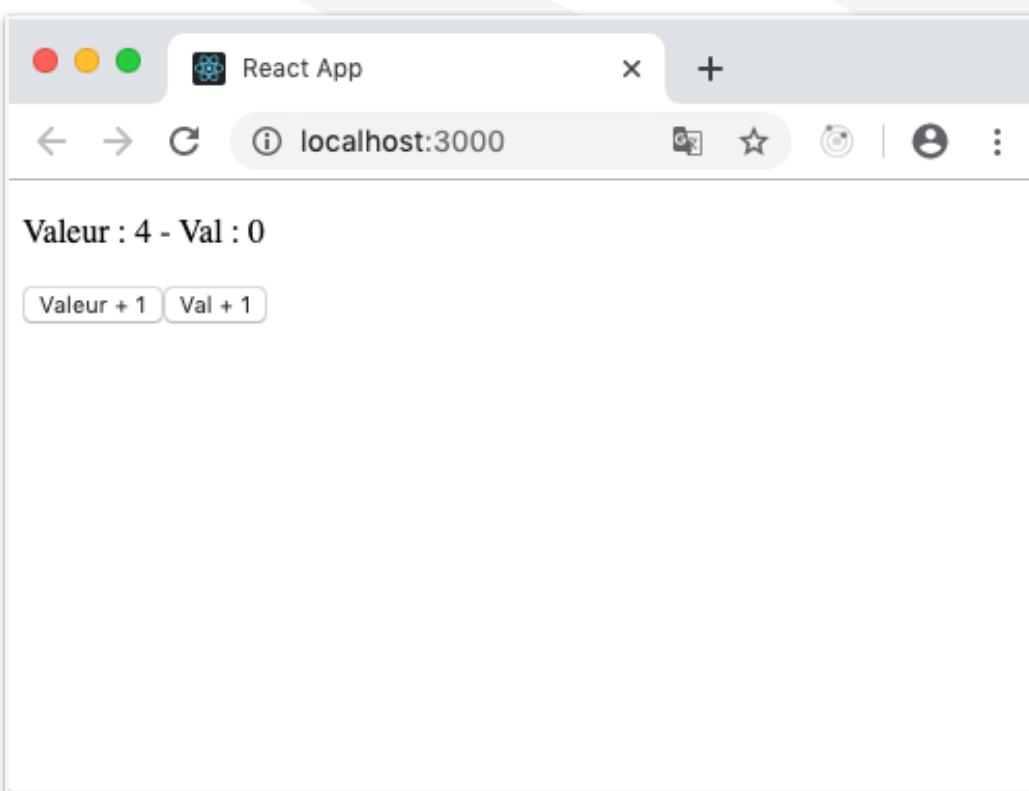


# NOTION D'ÉTAT LOCAL

- > Chaque composant dispose d'un état local (**state**).
  - > L'intérêt de l'état local est que toute modification le concernant déclenche un nouveau rendu des parties concernées du composant.
  - > Une propriété "normale" (hors état) n'a pas cet effet.
  - > L'état s'initialise dans le constructeur.
- 
- > **Très important :**
- ▶ L'état local est à traiter comme un objet **immutable**.
  - ▶ Toute modification doit passer par la méthode spécifique **setState()**.
  - ▶ Toute modification directe n'aura pas l'effet escompté excepté la première fois dans le constructeur.

# NOTION D'ÉTAT LOCAL

- > Principe de l'exemple : un compteur à deux valeurs
- ▶ Une valeur est dans l'état local (valeur)
- ▶ L'autre valeur est une propriété "standard" de la classe JavaScript (val)
- ▶ Chacune a un bouton dédié pour en augmenter la valeur.



# NOTION D'ÉTAT LOCAL

> Le code :

/src/App.js 1/2

```
1 import React, { Component } from 'react';
2
3 class Compteur extends Component
4 {
5   constructor(props) {
6     super(props);
7     this.state = {
8       valeur : 0
9     }
10    this.val = 0;
11  }
12
13  augmenterValeur = () => {
14    this.setState({valeur:this.state.valeur+1});
15  }
16
17  augmenterVal = () => {
18    this.val++;
19    console.log("Val : ",this.val);
20 }
```

The code is annotated with two callouts:

- A callout points to the line `this.state = { valeur : 0 }` with the text "Initialisation état local".
- A callout points to the line `this.setState({valeur:this.state.valeur+1})` with the text "Modification état local".

> Rappel : il faut toujours utiliser `setState()` pour modifier l'état local.

# NOTION D'ÉTAT LOCAL



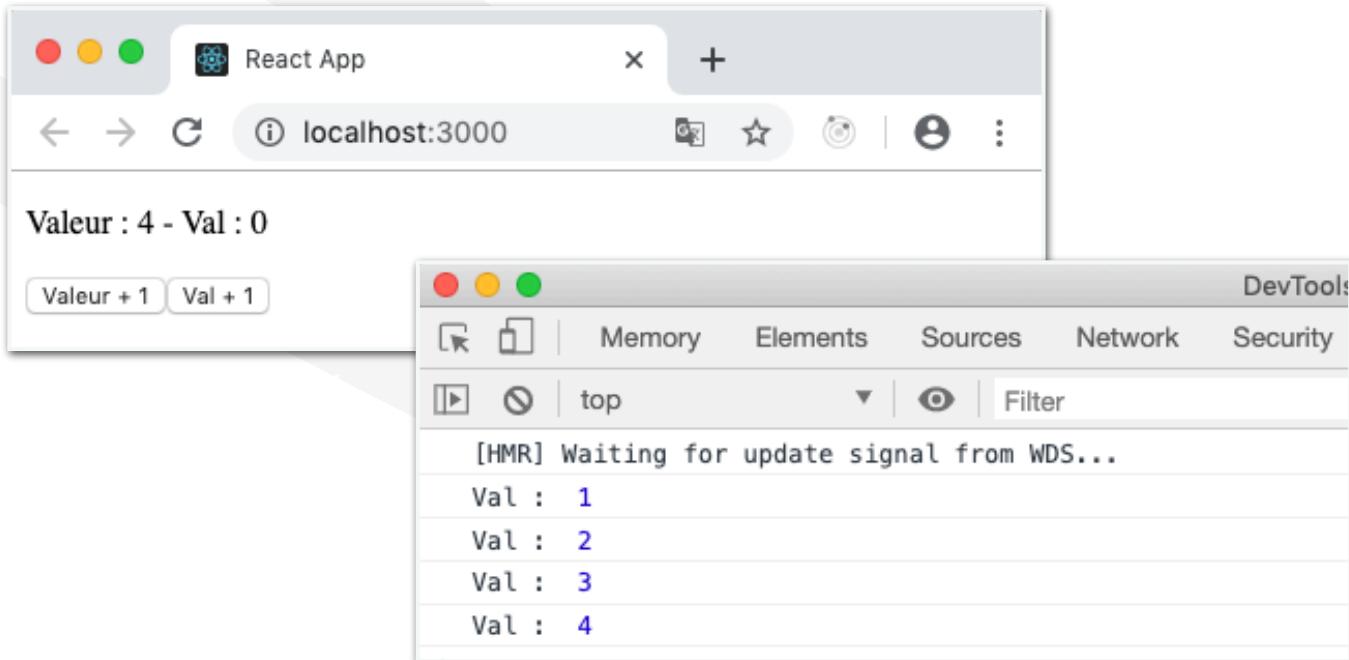
Le code :

/src/App.js 2/2

```
22  render() {
23    return (
24      <div>
25        <p>Valeur : {this.state.valeur} - Val : {this.val}</p>
26        <button onClick={this.augmenterValeur}>Valeur + 1</button>
27        <button onClick={this.augmenterVal}>Val + 1</button>
28      </div>;
29    }
30  }
31
32  const App = () => {
33    return (
34      <div>
35        <Compteur/>
36      </div>
37    )
38  }
39
40  export default App;
```

# NOTION D'ÉTAT LOCAL

> Résultat :



- > Le bouton "Valeur+1" modifie instantanément l'affichage par propagation de l'état local.
- > Le bouton "Val+1" n'a en revanche aucun effet visible, sauf dans la console où l'on voit que l'opération est bien effectuée mais que cela ne déclenche pas de mise à jour de l'affichage.

# TRANSMISSION AUX ENFANTS

- > Si une valeur de l'état local est utilisée comme props d'un enfant, la mise à jour se transmet.

/src/App.js

```
1 import React, { Component } from 'react';
2
3 class Compteur extends Component
4 {
5   constructor(props) {
6     super(props);
7     this.state = { valeur : 0 }
8   }
9
10  augmenterValeur = () => {
11    this.setState({valeur:this.state.valeur+1});
12  }
13
14  render() {
15    return (
16      <div>
17        <CompteurDisplay qqChose={this.state.valeur} />
18        <button onClick={this.augmenterValeur}>Valeur + 1</button>
19      </div>;
20    }
21  }
22
23 const CompteurDisplay = (props) => {
24   return <p>Valeur : {props.qqChose}</p>
25 }
```

Utilise une valeur du state comme props d'un enfant

Ajouter le code de <App/>

# STATE ET PROPS

- > Le state et les props sont deux notions clés de React.
- > Il faut garder à l'esprit 3 points importants :
  - ▶ Une modification du state ou d'une props déclenche automatiquement un nouveau rendu du composant concerné
  - ▶ Le state ne se modifie qu'au travers de **setState()**.
  - ▶ Une props ne se modifie pas. Elle se reçoit du parent.
- > Dans le fonctionnement interactif d'une application, il faut souvent modifier l'affichage pour refléter une modification des données (exemple données reçues en Ajax).
- > La force de React réside dans une certaine élégance dans la façon de le faire et aussi dans la philosophie normalement très modulaire des composants.
- > Évidemment tout n'est pas si simple...

# REACT

## partie 4



# COMMUNICATION ASCENDANTE

- > L'exemple qui suit a pour finalité d'expliquer la mise en place d'une communication ascendante, c'est-à-dire de faire remonter une information depuis un enfant (de son state) vers un son parent.
- > La mise en place de l'exemple va passer par un certain nombre d'étapes intermédiaires qui seront l'occasion de voir des pratiques courantes avec React.
- > Cet exemple va gérer plusieurs compteurs que nous allons commencer par mettre en place en tant que composant et l'utiliser.
- > Voici la structure du projet :

```
└─ src
    └─ App.js
    # Compteur.css
    └─ Compteur.js
    └─ index.js
    └─ serviceWorker.js
    └─ setupTests.js
```

# INTÉGRER DU CSS

- > Cette fois-ci nous n'allons pas nous contenter de l'apparence par défaut du HTML.
- > Nous allons donner un peu de style à nos compteurs.

/src/Compteur.css

```
1 .compteur {  
2     display:inline-block;  
3     width: 200px;  
4     text-align: center;  
5     margin:4px;  
6     border:1px solid black;  
7     padding: 5px;  
8 }  
9  
10 .compteur button {  
11     min-width:50px;  
12 }
```

- > Rien de neuf à ce niveau.

# COMPOSANT COMPTEUR AVEC CSS

> Voici le code du compteur.

/src/Compteur.js

```
1 import React, {Component} from "react";
2 import './Compteur.css'; ← Intégration du fichier CSS
3
4 class Compteur extends Component {
5     constructor(props) {
6         super(props);
7         this.state = { valeur : 0};
8     }
9
10    btnClickHandler(ajout) {
11        this.setState({ valeur: this.state.valeur+ajout });
12    }
13
14    render() {
15        return (<div className="compteur">
16            <p>{this.state.valeur}</p>
17            <button onClick={this.btnClickHandler.bind(this,1)}>+1</button>
18            <button onClick={this.btnClickHandler.bind(this,-1)}>-1</button>
19        </div>);
20    }
21 }
22
23 export default Compteur;
```

Attention c'est className et  
non class !

> Noter le **bind()** avec plusieurs valeurs lignes 17 et 18. Permet de gérer le +1 / -1 avec un seul gestionnaire.

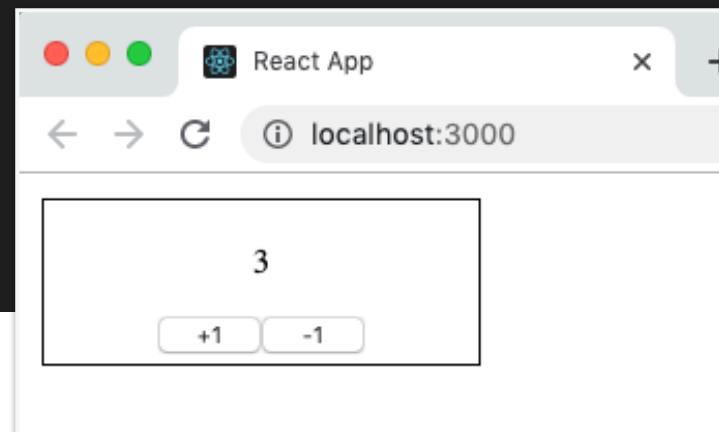
# COMPOSANT COMPTEUR AVEC CSS

> Voici le code du compteur

/src/App.js

```
1 import React, { Component } from 'react';
2 import Compteur from "./Compteur";
3
4
5 class App extends Component
6 {
7   render() {
8     return (<div>
9       <Compteur/>
10    </div>);
11  }
12
13 }
14
15 export default App;
```

Import du composant  
<Compteur/>



# FAIRE UNE BOUCLE

- > Les interfaces d'application sont remplies d'éléments répétés.
  - > La construction par programmation implique une boucle.
  - > Problème, JSX n'autorise que l'insertion d'expression, il faut donc quelque chose qui retourne ce que l'on doit insérer comme résultat.
  - > La pratique courante se fait autour de la méthode **Array .map()** de JavaScript.
- 
- > Dans l'étape qui vient, nous allons ajouter la possibilité d'ajouter dynamiquement de nouveaux compteurs.
  - > Nous allons tricher un peu en construisant un tableau de 0 pour supporter la répétition avec **Array .map()**.
  - > Pour que cela se mette à jour automatiquement, il faut que notre tableau fasse partie du state. Il nous faut donc rajouter un constructeur, un gestionnaire d'événement, un bouton et une boucle.

# FAIRE UNE BOUCLE



Voici le code.

/src/App.js (1/2)

```
1  import React, { Component } from 'react';
2  import Compteur from "./Compteur";
3
4
5  class App extends Component
6  {
7    constructor(props) {
8      super(props);
9      this.state = {
10        compteurListe : [0]
11      }
12    }
13
14    ajouterCompteur = () => {
15      this.setState({
16        compteurListe : [...this.state.compteurListe,0]
17      })
18    }

```

The diagram shows two annotations with arrows pointing to specific lines of code:

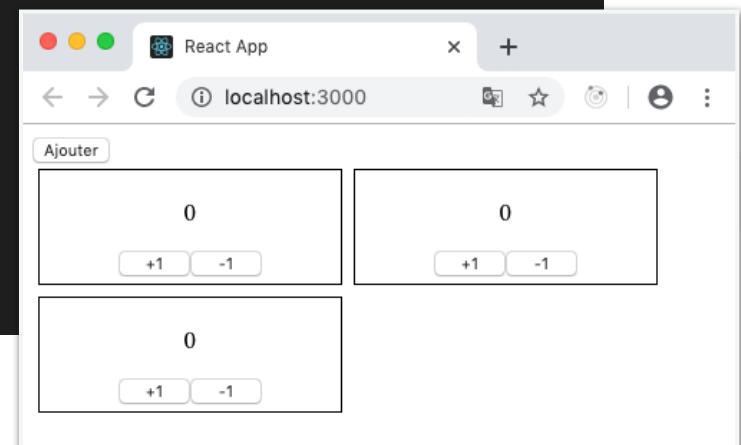
- An annotation with a vertical line and an arrow pointing to line 10: "Prévoit un compteur au démarrage".
- An annotation with a vertical line and an arrow pointing to line 16: "Ajoute un 0 au tableau".

# FAIRE UNE BOUCLE

> Voici le code.

/src/App.js (2/2)

```
20  render() {
21    return (<div>
22      <div>
23        <button onClick={this.ajouterCompteur}>Ajouter</button>
24      </div>
25      {
26        this.state.compteurListe.map((e,index)=>{ ←
27          return <Compteur key={index}/>
28        })
29      }
30    </div>);
31  }
32
33 }
34
35 export default App;
```



- > **Très important** : la props **key** est nécessaire pour toute répétition d'élément, cela permet d'aider React à optimiser ses traitements. Sans, il y aura un warning dans la console du navigateur.
- > La valeur à fournir peut être un vrai id.

# MAINTENIR UN TOTAL DES COMPTEURS

- > L'objectif est de maintenir la somme des valeurs indiquées par les compteurs.
- > La somme ne sera pas réellement calculée, d'une certaine manière l'opération conduite sur un compteur sera reproduite sur le total. Ce qui est plus facile.
  
- > Trois choses importantes sont à savoir dans ce cadre :
  - ▶ L'opération `setState()` est asynchrone, son résultat n'est pas nécessairement immédiat. La ligne "en-dessous" dans le code peut être exécutée avant la mise à jour effective de l'état.
  - ▶ `setState()` prévoit un second paramètre, un callback appelé une fois la mise à jour effectuée.
  - ▶ **Un parent ne peut pas accéder au contenu de ses enfants et réciproquement.**
  
- > Ce dernier point est un des "aspects particuliers" de React.

# COMMUNICATION ASCENDANTE

- > Alors comment faire ?
- > La technique usuelle consiste à faire que le parent transmette en tant que prop une de ses méthodes à ses enfants pour qu'ils puissent déclencher un traitement sur celui-ci.
- > Dans notre cas <App/> transmet une de ses méthodes aux <Compteur/>

/src/App.js (1/2)

```
1  import React, { Component } from 'react';
2  import Compteur from "./Compteur";
3
4
5  class App extends Component
6  {
7    constructor(props) {
8      super(props);
9      this.state = {
10        compteurListe : [0],
11        total : 0
12      }
13    }
14
15    ajouterCompteur = () => {
16      this.setState({
17        compteurListe : [...this.state.compteurListe,0]
18      })
19    }
}
```

# COMMUNICATION ASCENDANTE

- > La transmission de la méthode se fait ligne 35

/src/App.js (2/2)

```
21  updateTotal = (val) => {
22    this.setState({
23      total : this.state.total + val
24    })
25  }
26
27  render() {
28    return (<div>
29      <div>
30        <h1>{this.state.total}</h1>
31        <button onClick={this.ajouterCompteur}>Ajouter</button>
32      </div>
33      {
34        this.state.compteurListe.map((e,index)=>{
35          return <Compteur key={index}
36            onValeurChange={this.updateTotal} />
37        })
38      }
39    </div>);
40  }
41
42 }
43
44 export default App;
```

Choix de ne pas nommer la prop comme la méthode

# COMMUNICATION ASCENDANTE

> La transmission de la méthode se fait ligne 35

/src/Compteur.js

```
4  class Compteur extends Component {
5      constructor(props) {
6          super(props);
7          this.state = { valeur : 0 };
8      }
9
10     btnClickHandler(ajout) {
11         this.setState({ valeur: this.state.valeur+ajout }, () => {
12             this.props.onValeurChange(ajout);
13         });
14     }
15
16     render() {
17         return (<div className="compteur">
18             <p>{this.state.valeur}</p>
19             <button onClick={this.btnClickHandler.bind(this,1)}>+1</button>
20             <button onClick={this.btnClickHandler.bind(this,-1)}>-1</button>
21         </div>);
22     }
23 }
```

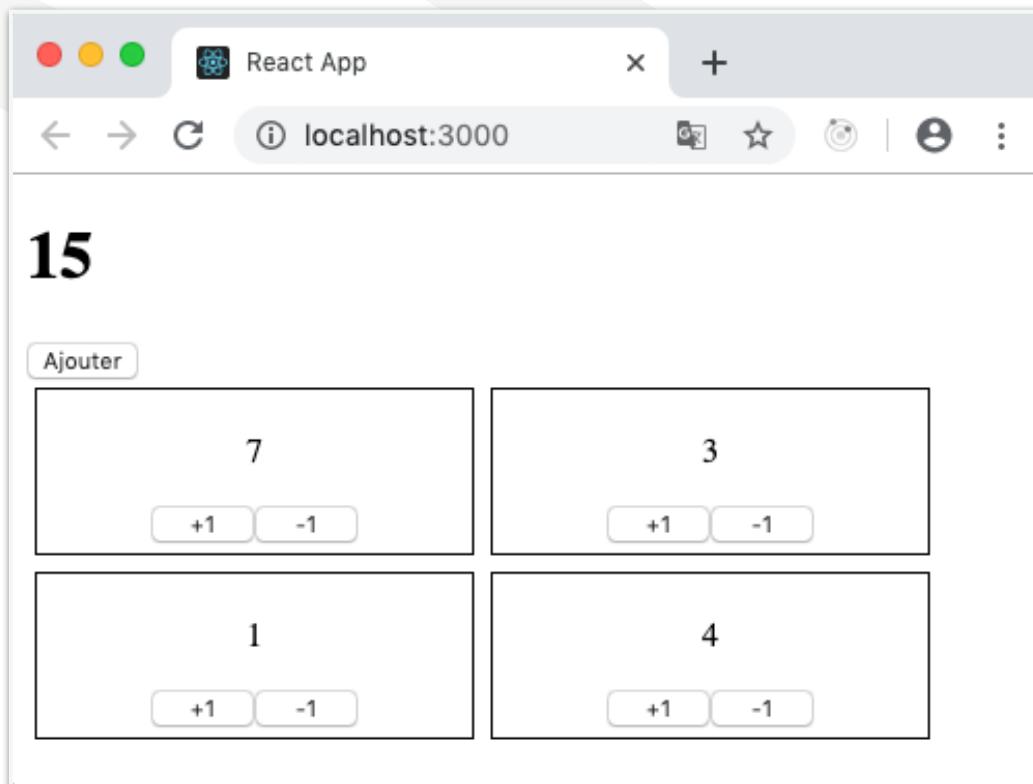
Pour faire un appel synchrone

Appel de la méthode de <App/> à partir de la prop

# COMMUNICATION ASCENDANTE



Résultat dans le navigateur



# REACT partie 5



# REACT partie 5.1



# PROJET PROFS : INTRODUCTION

- > L'objectif de cette partie est de construire une application React "du monde réel" qui permet de réaliser un CRUD sur les profs (nom, prénom, bureau).
- > La réalisation de cette application React suppose la disponibilité d'une API REST fonctionnelle pour assurer la persistance des données.
- > Au travers de cet exemple un peu plus conséquent nous allons à la fois :
  - ▶ Réinvestir certains concepts vus dans la première partie
  - ▶ Aborder de nouveaux concepts

Nom	Prénom	Bureau	Commandes
Chervy	Benjamin	218	<a href="#">Editer</a> <a href="#">Supprimer</a>
Ruchaud	William	210	<a href="#">Editer</a> <a href="#">Supprimer</a>
Gaudin	Hervé	217	<a href="#">Editer</a> <a href="#">Supprimer</a>

# PROJET PROFS : TOUR D'HORIZON

- > Voici la liste des caractéristiques principales de l'application :
  - ▶ Utilisation des props des states
  - ▶ Composants sous forme de classe ou de fonction
  - ▶ Utilisation de la bibliothèque Axios pour accéder à l'API Rest
  - ▶ Utilisation de Bootstrap pour le style de l'interface
  - ▶ Gestion des événements
  - ▶ Utilisation des formulaires "simples"
  - ▶ Utilisation de la bibliothèque react-router pour mettre en place une SPA avec plusieurs vues
  - ▶ Utilisation de la communication ascendante
  
- > Certaines points ont été volontairement traités de façons différentes pour montrer différentes manières de faire. Cela entraîne un manque d'homogénéité.

# API NÉCESSAIRE

- > Les routes suivantes sont nécessaires pour développer l'application

**GET** /api/v1/profs/ /api/v1/profs/

**POST** /api/v1/profs/ /api/v1/profs/

**GET** /api/v1/profs/{id} /api/v1/profs/{id}

**DELETE** /api/v1/profs/{id} /api/v1/profs/{id}

**PUT** /api/v1/profs/{id} /api/v1/profs/{id}

- > Les CORS doivent être configurés pour faire en sorte que l'application React puisse se fournir en données.
- > Prévoir des données exemples pour initier le développement.
- > Une API provisoire avec json-server peut être mise en place.

# PRÉPARER LE PROJET

- > Créer le projet avec Yarn

```
1 yarn create react-app prof-react-front
```

- > Se placer dans le dossier **prof-react-front**
- > Ajouter les dépendances suivantes avec **yarn** ou **npm** :

- ▶ axios
- ▶ bootstrap
- ▶ react-router-dom

```
5 "dependencies": {  
6   "@testing-library/jest-dom": "^5.11.4",  
7   "@testing-library/react": "^11.1.0",  
8   "@testing-library/user-event": "^12.1.10",  
9   "axios": "^0.21.0",  
10  "bootstrap": "^4.5.3",  
11  "react": "^17.0.1",  
12  "react-dom": "^17.0.1",  
13  "react-router-dom": "^5.2.0",  
14  "react-scripts": "4.0.1",  
15  "web-vitals": "^0.2.4"  
16},
```

- > L'extrait du fichier **package.json** précise les versions utilisées pour ce support.
- > Démarrer le projet avec **yarn start**

# PRÉPARER LE PROJET

- > Supprimer les fichiers suivants :
  - ▶ `/src/App.css`
  - ▶ `/src/App.test.js`
  - ▶ `/src/index.css` (supprimer la référence dans index.js)
  - ▶ `/src/logo.svg`
  - ▶ `/src/setupTest.js`

# PRÉPARER LE PROJET

- > Corriger le fichier `/src/index.js` pour ajouter Bootstrap.
- > Attention un `import` est supprimé par rapport à la version initiale.

`/src/index.js`

```
1 import React from 'react';
2 import ReactDOM from 'react-dom';
3 import App from './App';
4 import reportWebVitals from './reportWebVitals';
5 import 'bootstrap/dist/css/bootstrap.min.css';
6
7 ReactDOM.render(
8   <React.StrictMode>
9     <App />
10    </React.StrictMode>,
11    document.getElementById('root')
12  );
13
14 // If you want to start measuring performance in your app, pass a function
15 // to log results (for example: reportWebVitals(console.log))
16 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
17 reportWebVitals();
18
```



Intégration de Bootstrap

# PRÉPARER LE PROJET

- > Vider le contenu de /src/App.js sauf une balise <div></div>

/src/App.js

```
1  function App() {  
2  
3      return (  
4          <div>  
5              </div>  
6      );  
7  }  
8  
9  export default App;  
10
```

- > Le navigateur doit afficher une page blanche.

# MISE EN ROUTE BOOTSTRAP

- > Un simple code permet de s'assurer que Bootstrap est fonctionnel.
- > Rappel : en JSX le mot **class** est interdit dans une balise, il faut utiliser **className**.

/src/App.js

```
1 import React from 'react';
2
3 function App() {
4   return (
5     <div className="container">
6       <h1>Les Profs de l'école</h1>
7     </div>
8   );
9 }
10
11 export default App;
```



# REACT partie 5.2



# REACT-ROUTER

- > React Router ne fait pas partie de React.
- > C'est une bibliothèque qui permet d'utiliser de multiples routes à l'intérieur d'une application React.
- > Contrairement au HTML ou PHP de base, ici les routes ne vont pas désigner des fichiers, mais seulement indiquer à l'application React quel composant actionner en fonction de l'URL.
- > Techniquement, nous restons constamment sur le même fichier.
- > Nous n'allons aborder que le fonctionnement de base qui nous suffira pour réaliser notre application. Comme à chaque fois, la bibliothèque est beaucoup plus riche.

# REACT-ROUTER

- > Créer le fichier `/src/Home.js`

`/src/Home.js`

```
1 import { Link } from "react-router-dom";
2
3 function Home() {
4     const time = new Date().toLocaleString();
5     return <div>
6         <h2>Home</h2>
7         <p>{time}</p>
8         <Link to="/prof/90">Lien</Link>
9     </div>
10 }
11
12 export default Home;
```

- > Le composant `Link` permet de créer des liens pour React Router.
- > Ce composant inclue provisoirement l'affichage de l'heure pour que l'on se rende compte de l'instant où il est monté (affiché en jargon React).

# REACT-ROUTER

- > Créer le fichier /src/Form.js

/src/Form.js

```
1  function Form() {  
2      const time = new Date().toLocaleString();  
3      return <div>  
4          <h2>Form</h2>  
5          <p>{time}</p>  
6      </div>  
7  }  
8  
9  export default Form;
```

- > Ce composant est très similaire au précédent (attention aux lignes 1, 4 et 9).
- > Il ne possède pas de composant Link.

# REACT-ROUTER

- > Modifier le fichier /src/App.js

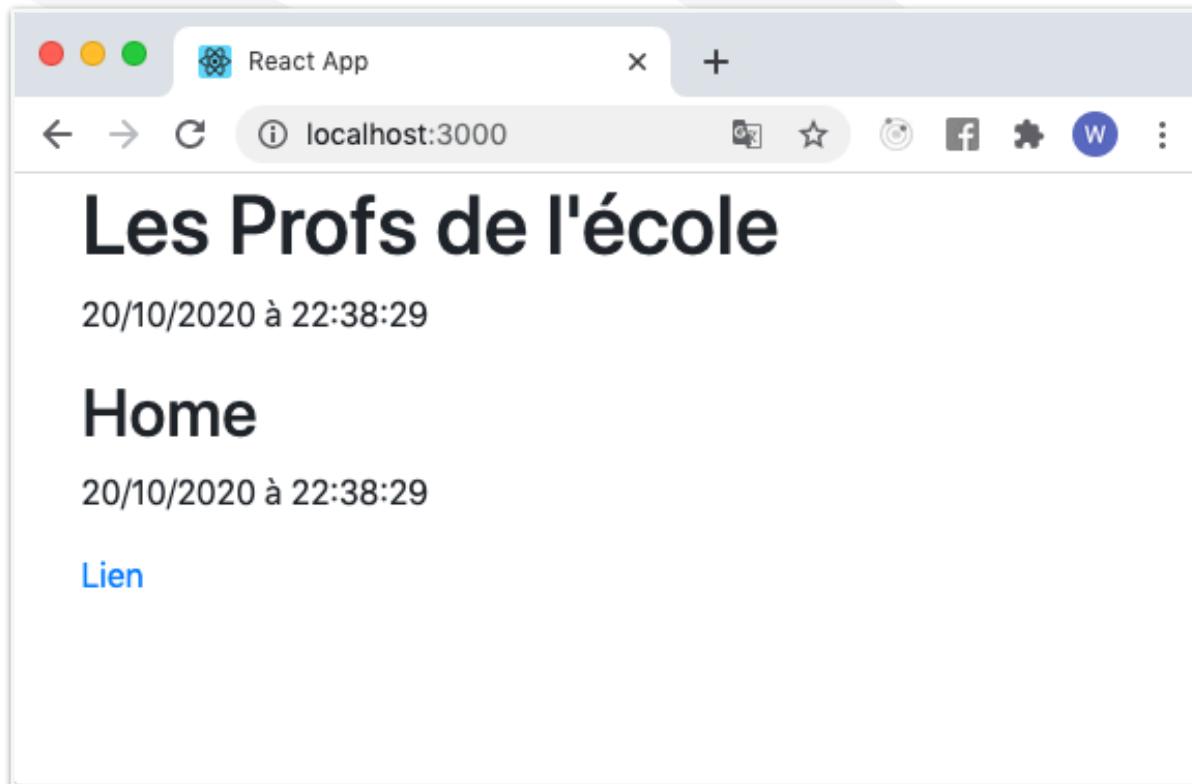
/src/App.js

```
1 import { BrowserRouter as Router, Route } from "react-router-dom";
2 import Home from "./Home";
3 import Form from "./Form";
4
5 function App() {
6   const time = new Date().toLocaleString();
7   return (
8     <div className="container">
9       <h1>Les profs de l'école</h1>
10      <p>{time}</p>
11      <Router>
12        <Route path="/" exact component={Home}/>
13        <Route path="/prof/:id" component={Form}/>
14        <Route path="/prof-create" component={Form}/>
15      </Router>
16    </div>
17  );
18}
19
20 export default App;
21
```

- > Test sur la page suivante

# REACT-ROUTER

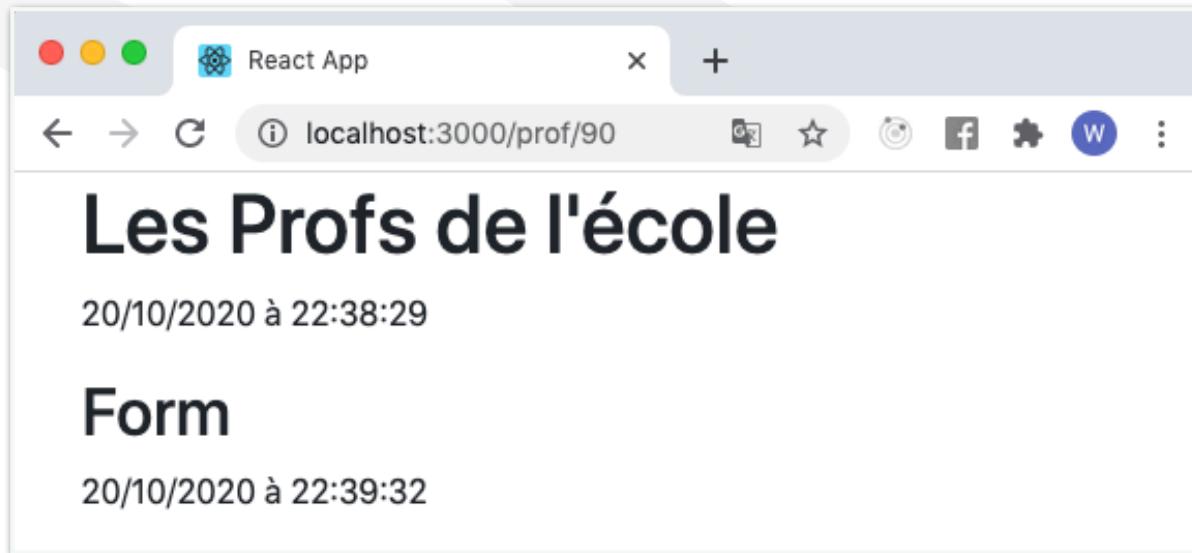
- > Observer les heures au lancement de votre application, elles sont identiques.



- > Patienter quelques secondes puis cliquer sur Lien. Le résultat est sur la page suivante.

# REACT-ROUTER

- > Il y a eu changement dans la partie "basse" : Form à remplacé Home.



- > L'heure du haut n'a pas bougé, celle du bas est différente.
- > React procède par montage / démontage de composants.
- ▶ Le composant Home initialement monté est démonté.
- ▶ Le composant Form est monté à sa place.

# REACT-ROUTER

- > Pourquoi avoir trois routes ?
  - ▶ La route `/` affichera la liste des profs.
  - ▶ La route `/prof/:id` permettra l'édition d'un prof
  - ▶ La route `/prof-create` permettra la création d'un prof
  
- > Les deux dernières routes font appel au même composant Form.
- > Pour que cela fonctionne, nous allons utiliser une prop : **edit** qui vaudra **true** ou **false** selon qu'il faut utiliser Form en mode édition ou création.
  
- > Il nous faut changer l'écriture des route pour que cela fonctionne, au lieu d'utiliser la prop **component** (utiliser un composant tel quel) nous allons utiliser la prop **render** qui permet de passer une fonction de rendu.

# REACT-ROUTER

- > Modifier le fichier /src/App.js

/src/App.js

```
1 import { BrowserRouter as Router, Route } from "react-router-dom";
2 import Home from "./Home";
3 import Form from "./Form";
4
5 function App() {
6   const time = new Date().toLocaleString();
7   return (
8     <div className="container">
9       <h1>Les profs de l'école</h1>
10      <p>{time}</p>
11      <Router>
12        <Route path="/" exact component={Home}/>
13        <Route path="/prof/:id" render={(props) => <Form edit={true} {...props}>
14          </> } />
15        <Route path="/prof-create" render={(props) => <Form edit={false} {...props}>
16          </> } />
17      </Router>
18    </div>
19  );
20
21 export default App;
```

- > Il n'y a aucun retour à la ligne, seulement la capture qui est trop étroite.

# REACT-ROUTER

- > Regardons le détail de la route.

```
13 |     <Route path="/prof/:id" render={({props) => <Form edit={true} {...props}>
|       /> } />
```

- > Le principe est une fonction flèche qui retourne un composant : `(props) => <Form />`
- > Il nous faut notre prop `edit` pour valoir `true` ou `false`
- > Il nous faut aussi récupérer les props "invisibles" qui sont transmises par React et React Router. C'est pour cela que nous les transmettons par destructuration `{...props}`

# REACT-ROUTER

- > Modifier le code de /src/Form.js

/src/Form.js

```
1 | function Form(props) {
2 |   return <div>
3 |     <h2>Form</h2>
4 |     <p>{JSON.stringify(props)}</p>
5 |   </div>
6 |
7 |
8 | export default Form;
```

## Form

```
{"edit":true,"history":{"length":48,"action":"POP","location":
{"pathname":"/prof/90","search":"","hash":"","key":"rohmf8"}}, "location":
 {"pathname":"/prof/90","search":"","hash":"","key":"rohmf8"}, "match":
 {"path":"/prof/:id","url":"/prof/90","isExact":true,"params":{"id":"90"}}}
```

- > Bien penser à ajouter **props** à la première ligne !
- > Nous utilisons ici, une façon de debuger les props : en faire un affichage JSON.
- > Aller à la route /prof/45 puis /prof-create la prop **edit** doit valoir respectivement **true** puis **false**.
- > Cela nous donne aussi l'opportunité de voir comment récupérer le paramètre **id** de la route : **props.match.params.id**.

# REACT partie 5.3



# PROFSERVICE

- > Histoire de regrouper tous nos appels à l'API REST, nous allons créer un nouveau fichier : `/src/ProfService.js`
- > Voici sa version de base.

```
/src/ProfService.js
1 import axios from 'axios'
2
3 const rootURL = 'http://localhost:8080/api/v1';
4
5 class ProfService {
6     getAll() {
7         return axios.get(rootURL + "/profs");
8     }
9 }
10
11 export default new ProfService();
```

- > En regroupant tous les appels axios ici, en cas de modification de l'API à utiliser, tout sera au même endroit, et l'URL de base n'est écrite qu'une seule fois, ce qui est plus propre.
- > `axios.get()` retourne une promesse.

- > Home va subir une grande transformation.
- > La gestion des données asynchrone, implique l'utilisation du state et donc le passage à une classe.
- > Nous verrons plus tard qu'il existe une autre manière de faire.
- > Rappels :
  - ▶ Le state s'initie dans le constructeur.
  - ▶ Le constructeur doit impérativement commencer par super(props)
- > Les composants React ont un **cycle de vie**. Ce cycle de vie se manifestera pour nous sous la forme d'une méthode héritée : `componentDidMount()` qui est déclenchée en fin de montage du composant.
- > C'est l'endroit préconisé pour déclencher le chargement des données.



## La transformation est importante

/src/Home.js

```
1 import { Component } from "react";           ← Pour utiliser Component
2 import { Link } from "react-router-dom";
3 import ProfService from "./ProfService";
4
5 class Home extends Component
6 {
7     constructor(props) { ← Mise en place du state vide
8         super(props);
9         this.state = { profSet : [] };
10    }
11
12    componentDidMount() {
13        ProfService.getAll()
14            .then(response => {
15                this.setState({profSet : response.data.data })
16            })
17    }
18
19    render() {
20        return <div>
21            <h2>Home</h2>
22            <p>{JSON.stringify(this.state)}</p>
23        </div>
24    }
25}
26 export default Home;
```

Rappel, le state se manipule avec setState()

Pour voir le contenu de ce qui est reçu



Le résultat doit ressembler à la capture ci-dessous (avec vos données à vous).

React App

localhost:3000

# Les Profs de l'école

20/10/2020 à 23:15:46

## Home

```
{"profSet": [{"id":10,"nom":"Chervy","prenom":"Benjamin","bureau":218,"motdepasse":"$2b$10-20T12:54:21.000Z","updatedAt":"2020-10-20T12:54:21.000Z"}, {"id":12,"nom":"Ruchaud","prenom":"William","bureau":210,"motdepasse":"$2b$010-20T17:56:14.000Z","updatedAt":"2020-10-20T17:56:14.000Z"}, {"id":14,"nom":"Gaudin","prenom":"Hervé","bureau":217,"motdepasse":"$2b$05$10-20T18:12:05.000Z","updatedAt":"2020-10-20T18:12:05.000Z"}]}
```

- > Pour afficher les données, nous allons utiliser un tableau Bootstrap.
- > Nous allons le faire à l'aide d'un sous-composant : **Prof** qui prendra en charge l'affichage d'une ligne (<tr>).
- > Le composant **Prof** sera sous la forme d'une fonction et ne sera pas exporté.
- > Le code étant volumineux, il est divisé en deux captures.



## La transformation est importante

/src/Home.js (1/2)

```
19    render() {
20        return <div>
21            <div className="text-right">
22                <Link to="/prof-create">
23                    <button className="btn-primary">Ajouter</button>
24                </Link>
25            </div>
26            <div>
27                <table className="table table-condensed table-hover">
28                    <thead>
29                        <tr>
30                            <th>Nom</th>
31                            <th>Prénom</th>
32                            <th>Bureau</th>
33                            <th>Commandes</th>
34                        </tr>
35                    </thead>
36                    <tbody>
37                        {
38                            this.state.profSet.map((item,index) => <Prof prof={item}>
39                                key={'prof'+index} />)
40                        }
41                    </tbody>
42                </table>
43            </div>
44        }
```

Lien pour ajouter

Utilisation de map pour boucler sur un tableau

Obligatoire pour une boucle !!



## La transformation est importante

/src/Home.js (2/2)

```
47  function Prof(props) {
48    const { id, nom, prenom, bureau } = props.prof;
49
50    return <tr>
51      <td>{nom}</td>
52      <td>{prenom}</td>
53      <td>{bureau}</td>
54      <td>
55        <Link to={"prof/"+id}><button className="btn">Editer</button></Link>
56        <button className="btn">Supprimer</button>
57      </td>
58    </tr>
59  }
```

Déstructuration pour éviter de répéter "prof."



Résultat :

The screenshot shows a web browser window titled "React App" at "localhost:3000". The page is titled "Les Profs de l'école" and displays a timestamp "20/10/2020 à 23:29:08". A blue button labeled "Ajouter" is visible. Below is a table with three rows:

Nom	Prénom	Bureau	Commandes
Chervy	Benjamin	218	<a href="#">Editer</a> <a href="#">Supprimer</a>
Ruchaud	William	210	<a href="#">Editer</a> <a href="#">Supprimer</a>
Gaudin	Hervé	217	<a href="#">Editer</a> <a href="#">Supprimer</a>



Cliquer sur Editer ou sur Ajouter doit fonctionner et arriver sur le composant Form.

# REACT partie 5.4



# MISE EN PLACE DU DELETE

> Dans le fichier `/src/ProfService.js`.

`/src/ProfService.js`

```
1 import axios from "axios";
2
3 const rootURL = 'http://localhost:8080/api/v1';
4
5 class ProfService {
6
7     getAll() {
8         return axios.get(rootURL+'/profs');
9     }
10
11     remove(id) {
12         return axios.delete(rootURL+"/profs/"+id);
13     }
14 }
15
16 export default new ProfService();
```

# MISE EN PLACE DU DELETE

- > Dans `/src/Home.js` nous allons ajouter une fonction locale au composant `Prof` : `deleteHandler`

`/src/Home.js`

```
49  function Prof(props) {
50    const { id, nom, prenom, bureau } = props.prof;
51
52    function deleteHandler() {
53      ProfService.remove(id);
54    }
55
56    return <tr>
57      <td>{nom}</td>
58      <td>{prenom}</td>
59      <td>{bureau}</td>
60      <td>
61        <Link to={"prof/"+id}><button className="btn">Editer</button></Link>
62        <button className="btn" onClick={deleteHandler}>Supprimer</button>
63      </td>
64    </tr>
65  }
```

# MISE EN PLACE DU DELETE

- > Ça marche, mais pas comme on peut le penser naïvement.
- > Si on clique sur le bouton Supprimer d'un prof, puis qu'on rafraîchit le navigateur, le prof a bien disparu;
- > Pourquoi ne disparait-il pas automatiquement de la liste ?
- > C'est un prof, c'est plutôt tenace...
- > Humour mis à part, il ne disparait pas parce que rien n'a changé au niveau du composant Home. Ce n'est pas parce qu'on supprime un élément sur une API REST que toute la Terre est prévenue et surtout pas nos composants.
- > Deux pistes :
  - ▶ Recharger toute la liste : peut être coûteux
  - ▶ Retirer l'élément supprimé si on est sûr qu'il l'est : moins coûteux

# MISE EN PLACE DU DELETE

- > OK on va retirer de la liste.
  - > Question piège : qui effectue la suppression ? Réponse : notre composant Prof.
  - > Les créateurs de React ne semblent pas avoir prévu le hara-kiri des composants, donc c'est le composant "au dessus" de procéder.
- 
- > Nous avons déjà vu ce qui va nous servir à le faire : la communication ascendante;
  - > Home va mettre en place une fonction qu'il transmettra à Prof pour qu'il puisse déclencher la suppression "visuelle" (parce que la suppression BDD est déjà faite).

# MISE EN PLACE DU DELETE

- > Ajouter la méthode `removeProf()` :

/src/Home.js

```
13  componentDidMount() {
14    ProfService.getAll()
15    .then(response => {
16      this.setState({profSet : response.data.data })
17    })
18  }
19
20  removeProf(id) {
21    const res = this.state.profSet.filter((item) => item.id !==id );
22    this.setState({ profSet : res })
23  }
24
25  render() {
```

- > Pour retirer un élément de la liste en mémoire, on la filtre en ne gardant que les profs qu'il ne faut pas effacer.

# MISE EN PLACE DU DELETE

- > Au moment du .map() il faut "donner" méthode `removeProf()` (prop `onDelete`).

/src/Home.js

```
42          <tbody>
43          {
44              this.state.profSet.map((item,index) => <Prof
45                  prof={item}
46                  key={'prof'+index}
47                  onDelete={ this.removeProf }
48              />)
49          }
50      </tbody>
```

- > Puis faire l'appel dans Prof :

/src/Home.js

```
57  function Prof(props) {
58      const { id, nom, prenom, bureau } = props.prof;
59
60      function deleteHandler() {
61          ProfService.remove(id)
62              .then(() => {
63                  props.onDelete(id);
64              });
65      }
```

- > **IMPORTANT** : Tester une suppression et lire la page suivante

# MISE EN PLACE DU DELETE

> Sauf que (volontaire) :

The screenshot shows a browser window titled "React App" at "localhost:3000". The main content area displays a red error message: "Unhandled Rejection (TypeError): Cannot read property 'profSet' of undefined". Below the error message, the code for the "deleteProf" function is shown:

```
deleteProf [as onDelete]
src/Home.js:21

18 | }
19 |
20 | deleteProf(id) {
> 21 |   const res = this.state.profSet.filter((item) => item.id !== id);
  22 | ^  this.setState({profSet:res});
23 |
24 | }
```

Below this, there is a link "View compiled". Further down, another code snippet for an anonymous function is shown:

```
(anonymous function)
src/Home.js:57

54 | function deleteHandler() {
55 |   ProfService.delete(id)
56 |   .then(() => {
> 57 |     props.onDelete(id);
  58 |   })
59 |
60 | }
```

Below this, there is another link "View compiled". At the bottom of the error screen, a note states: "This screen is visible only in development. It will not appear if the app crashes in production. Open your browser's developer console to further inspect this error. Click the 'X' or hit ESC to dismiss this message."

# MISE EN PLACE DU DELETE

- > Le message d'erreur alerte qu'on essaie de lire `profSet` sur quelque chose qui n'est pas défini.
- > La ligne désignée est celle du `filter()`, alors quoi ?
- > Alors qu'on a oublié qu'il faut faire un bind (volontaire aussi) !

/src/Home.js

```
1 import { Component } from "react";
2 import { Link } from "react-router-dom";
3 import ProfService from "./ProfService";
4
5 class Home extends Component
6 {
7     constructor(props) {
8         super(props);
9         this.state = { profSet : [] };
10    }
11    this.removeProf = this.removeProf.bind(this);
12 }
13
```

- > Maintenant le delete est terminé. Il nous reste à faire le add/edit.

# REACT partie 5.5



# ADD/EDIT

- > Avec le add/edit nous allons travailler avec les formulaires.
- > À mon goût, les formulaires ne sont pas la meilleure partie de React. Ils sont plein de subtilités (composants contrôlés, non contrôlés). Leur compréhension n'est pas immédiate.
  
- > Le challenge est le suivant : faire en sorte qu'un champ de formulaire (champ texte par exemple) ait sa valeur (**value**) dans son state.
- > Si on modifie le state, la valeur est automatiquement modifiée à l'affichage, et si on modifie la valeur en tapant quelque chose, le state est automatiquement modifié.
- > Cela paraît simple, sauf que se rajoute la valeur initiale qui, elle, est plutôt props;
- > Et c'est là que ça devient compliquer.
  
- > Dans les mauvaises pratiques, les "anti-patterns", il y a le fait de recopier les props dans le state.

# ADD/EDIT

- > Malgré tout, il va falloir faire avec. C'est pour cela que nous allons rester simple dans nos objectifs : permettre juste la saisie du nom, prénom et bureau sans ajouter de vérification.
- > Première chose à faire : passer le composant **Form** en classe.

/src/Form.js

```
1  import { Component } from "react";
2
3  class Form extends Component
4  {
5      constructor(props) {
6          super(props);
7      }
8
9      render() {
10         return <div>
11             <input type="text" id="nom" />
12             <button>Enregistrer</button>
13         </div>
14     }
15 }
16
17 export default Form;
```

# ADD/EDIT



Pour l'instant nous travaillons sans l'apparence Bootstrap



# ADD/EDIT

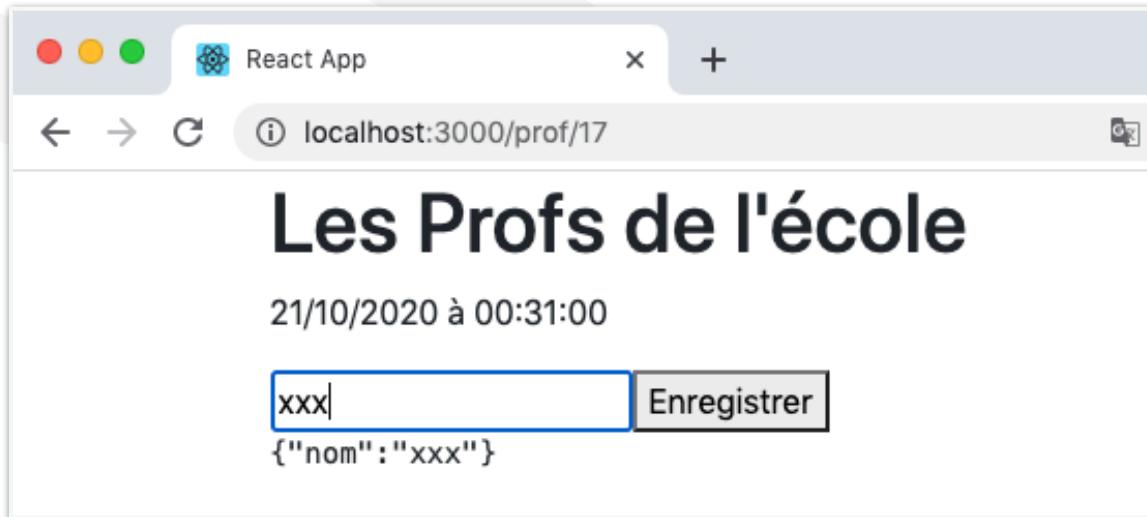
- > Comme dit précédemment, l'idée est de maintenir synchronisés le state de la classe **Form** et la prop **value** du composant **input**.
- > Il nous faut un state. Nous allons rajouter de quoi de debuger avec un **JSON.stringify()**.

/src/Form.js

```
1 import { Component } from "react";
2
3 class Form extends Component
4 {
5     constructor(props) {
6         super(props);
7         this.state = {
8             nom : "xxx"
9         }
10    }
11    render() {
12        return <div>
13            <input type="text" id="nom" value={this.state.nom} />
14            <button>Enregistrer</button>
15            <pre>{JSON.stringify(this.state)}</pre>
16        </div>
17    }
18}
19
20 export default Form;
```

# ADD/EDIT

- > Essayer de taper quelque chose dans la zone de saisie.



- > Ça ne marche pas ! La zone est en lecture seule.
- > Elle est en lecture seule parce qu'il manque un morceau : la modification du state à partir de ce qui est tapé dans le composant <**input**>.
- > Sinon, comme la prop **value** de <**input**> est synchronisée sur le state du formulaire et que lui est pour l'instant fixe, ça ne risque pas de bouger.

# ADD/EDIT

/src/Form.js

```
1  import { Component } from "react";
2
3  class Form extends Component
4  {
5      constructor(props) {
6          super(props);
7          this.state = {
8              nom : "xxx"
9          }
10
11         this.changeHandler = this.changeHandler.bind(this);
12     }
13
14     changeHandler(evt) {
15         this.setState( { nom : evt.target.value });
16     }
17
18     render() {
19         return <div>
20             <input type="text" id="nom" value={this.state.nom}
21                 onChange={this.changeHandler}
22             />
23             <button>Enregistrer</button>
24             <pre>{JSON.stringify(this.state)}</pre>
25         </div>
26     }
27 }
28
29 export default Form;
```

# ADD/EDIT

- > Voilà une des démarches pour gérer les champs de formulaire (de type contrôlé comme un `<input>` de type texte) :
  - ▶ Avoir une valeur dans le state
  - ▶ Injecter la valeur le state dans le `<input>`
  - ▶ Positionner un `onChange` pour sauvegarder les modifications dans le state.
  - ▶ Penser à faire le `bind` nécessaire pour que cela marche.
- > On ne peut pas dire que ce soit évident pour le néophyte !
- > Ce n'est qu'une façon de faire.
- > Cela veut dire qu'il faut le faire pour pratiquement chaque `<input>` du formulaire !
- > Heureusement, il est possible de se faciliter un tout petit peu la vie, en écrivant un `changeHandler` générique.

# ADD/EDIT

- > Par une petite modification du code de `changeHandler`, on peut l'utiliser pour plusieurs champ à la fois.

/src/Form.js

```
3  class Form extends Component {
4      constructor(props) {
5          super(props);
6          this.state = { nom : "xxx" }
7
8          this.changeHandler = this.changeHandler.bind(this);
9      }
10
11     changeHandler(evt) {
12         const fieldId = evt.target.id;
13         let resultingState = this.state;
14         resultingState[fieldId] = evt.target.value;
15         this.setState(resultingState);
16     }
17
18     render() {
```

- > Contrôler que la synchronisation formulaire/state fonctionne toujours.

# REACT partie 5.6



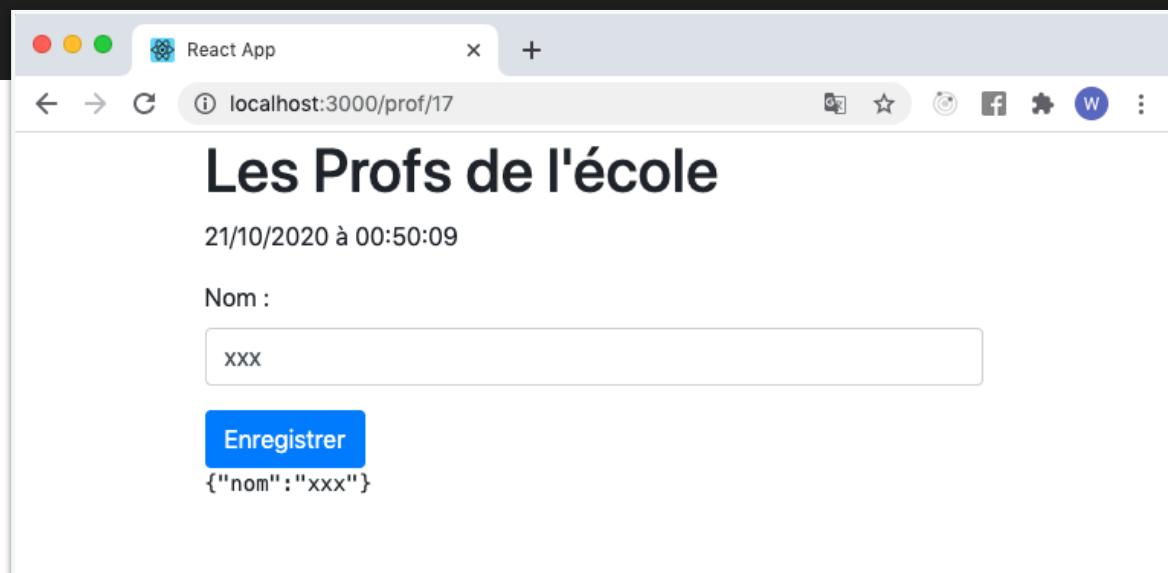
# MISE EN FORME BOOTSTRAP



Avant d'aller plus loin, nous allons mettre en forme l'existant en BootStrap

/src/Form.js

```
21     render() {
22       return <div>
23         <div className="form-group">
24           <label htmlFor="nom">Nom : </label>
25           <input type="text" id="nom" className="form-control"
26             value={this.state.nom}
27             onChange={this.changeHandler}>
28           />
29         </div>
30         <button className="btn btn-primary">Enregistrer</button>
31         <pre>JSON.stringify(this.state)</pre>
32       </div>
33     }
```



# AJOUTER LE PRENOM ET LE BUREAU

- > Pour ajouter le prenom et le bureau, il nous faut commencer par le constructeur pour les enregistrer dans le state.

/src/Form.js

```
1 import { Component } from "react";
2
3 class Form extends Component
4 {
5     constructor(props) {
6         super(props);
7         this.state = {
8             nom : "xxx",
9             prenom : "ooo",
10            bureau : "0"
11        }
12
13        this.changeHandler = this.changeHandler.bind(this);
14    }
```

# AJOUTER LE PRENOM ET LE BUREAU

- > Ensuite il suffit de les rajouter dans `render()`.
- > Attention aux erreurs de copier-coller ! Par bloc, il y a 4 endroit à modifier.

```
23   render() {  
24     return <div>  
25       <div className="form-group">  
26         <label htmlFor="nom">Nom : </label>  
27         <input type="text" id="nom" className="form-control"  
28           value={this.state.nom}  
29           onChange={this.changeHandler}  
30         />  
31       </div>  
32       <div className="form-group">  
33         <label htmlFor="prenom">Prénom : </label>  
34         <input type="text" id="prenom" className="form-control"  
35           value={this.state.prenom}  
36           onChange={this.changeHandler}  
37         />  
38       </div>  
39       <div className="form-group">  
40         <label htmlFor="bureau">Bureau : </label>  
41         <input type="text" id="bureau" className="form-control"  
42           value={this.state.bureau}  
43           onChange={this.changeHandler}  
44         />  
45       </div>  
46       <button className="btn btn-primary">Enregistrer</button>  
47       <pre>{JSON.stringify(this.state)}</pre>  
48     </div>  
49   }
```

# AJOUTER LE PRENOM ET LE BUREAU

- > Normalement tout doit fonctionner, taper quelque chose dans le prénom ou le bureau doit modifier le state.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/prof/17". The page has a title "Les Profs de l'école" and a timestamp "21/10/2020 à 00:55:11". It contains three input fields: "Nom :" with value "xxx", "Prénom :" with value "ooo", and "Bureau :" with value "210". A blue button labeled "Enregistrer" is at the bottom. Below the button, a JSON object is displayed: {"nom": "xxx", "prenom": "ooo", "bureau": "210"}.

```
{"nom": "xxx", "prenom": "ooo", "bureau": "210"}
```

# ON PEUT FAIRE MIEUX

- > Le côté répétitif est évident, avec le risque d'erreur qui s'accompagne.
- > On a tout intérêt à passer par un composant pour nous simplifier la tâche.
- > Nous allons mettre en place, dans le fichier **/src/Form.js** un composant fonction pour générer un bloc façon Bootstrap : **TextInput**.

/src/Form.js

```
35  function TextInput(props) {
36    return (
37      <div className="form-group">
38        <label htmlFor={props.id}>{props.label} : </label>
39        <input type="text" id={props.id} className="form-control"
40          value={props.value}
41          onChange={props.onChange}
42        />
43      </div>
44    )
45  }
```

- > Ce composant attend : un id, un label, une value et un onChange comme props.

# ON PEUT FAIRE MIEUX

- > Cela transforme radicalement la méthode `render()`.
- > C'est bien là toute la force de React, pouvoir facilement remplacer un code JSX/HTML répétitif par un composant plus concis.

/src/Form.js

```
33     render() {
34         return <div>
35             <TextInput id="nom" label="Nom"
36                 value={this.state.nom} onChange={this.changeHandler} />
37             <TextInput id="prenom" label="Prénom"
38                 value={this.state.prenom} onChange={this.changeHandler} />
39             <TextInput id="bureau" label="Bureau"
40                 value={this.state.bureau} onChange={this.changeHandler} />
41
42             <button className="btn btn-primary">Enregistrer</button>
43             <pre>{JSON.stringify(this.state)}</pre>
44         </div>
45     }
```

# REACT partie 5.7



# PROFSERVICE

> Nous allons terminer /src/ProfService.js

/src/ProfService.js

```
1 import axios from "axios";
2
3 const rootURL = 'http://localhost:8080/api/v1';
4
5 class ProfService {
6
7     getAll() {
8         return axios.get(rootURL+'/profs');
9     }
10
11    getById(id) {
12        return axios.get(rootURL+"/profs/"+id);
13    }
14
15    update(id,data) {
16        return axios.put(rootURL+"/profs/"+id,data);
17    }
18
19    create(data) {
20        return axios.post(rootURL+"/profs/",data);
21    }
22
23    remove(id) {
24        return axios.delete(rootURL+"/profs/"+id);
25    }
26}
27
28 export default new ProfService();
```

# CHARGEMENT

- > Nous devons procéder au chargement des données pour l'édition d'un prof.
- > Cela se fera avec **componentDidMount** qui est appelée quand le composant a fini de s'afficher.
- > À tester en édition ou en création.
- > À noter l'apparition du mot de passe (juste pour la sauvegarde, mais vide).

/src/Form.js

```
4  class Form extends Component
5  {
6      constructor(props) {
7          super(props);
8          this.state = {
9              nom : '',
10             prenom : '',
11             bureau : ''
12         }
13
14         this.changeHandler = this.changeHandler.bind(this);
15     }
16
17     componentDidMount() {
18         if(this.props.edit) {
19             ProfService.getById(this.props.match.params.id)
20                 .then((response) => { this.setState({ ...response.data.data }); })
21         }
22     }
}
```

# SAUVEGARDE

- > Pour déclencher l'enregistrement, il nous faut un gestionnaire d'événement sur le bouton Enregistrer, donc un **bind**, donc intervenir dans le constructeur.

/src/Form.js 1/3

```
18     saveHandler() {
19       if(this.props.edit) {
20         console.log('Update');
21         ProfService.update(this.props.match.params.id,this.state)
22           .then(() => {
23             this.props.history.push('/');
24           })
25       } else {
26         console.log('Create');
27         ProfService.create({...this.state,motdepasse:''})
28           .then(() => {
29             this.props.history.push('/');
30           })
31       }
32     }
```

# SAUVEGARDE

> Il n'y a plus qu'à l'appeler dans le `onClick`.

/src/Form.js 2/3

```
6  constructor(props) {
7    super(props);
8    this.state = {
9      nom : '',
10     prenom : '',
11     bureau : ''
12   }
13
14   this.changeHandler = this.changeHandler.bind(this);
15   this.saveHandler = this.saveHandler.bind(this);
16 }
```

/src/Form.js 3/3

```
44  render() {
45    return <div>
46      <TextInput id={"nom"} label={"Nom"} value={this.state.nom} onChange=
47        {this.changeHandler} />
48      <TextInput id={"prenom"} label={"Prénom"} value={this.state.prenom}
49        onChange={this.changeHandler} />
50      <TextInput id={"bureau"} label={"Bureau"} value={this.state.bureau}
51        onChange={this.changeHandler} />
52      <button className="btn btn-primary" onClick={this.saveHandler}>
53        >Enregistrer</button>
54      <pre>{JSON.stringify(this.state)}</pre>
55    </div>
56  }
```

# BOUTON ANNULER

> Dernière étape, pour le confort de l'utilisateur, nous allons rajouter un bouton pour revenir du formulaire vers la liste.

/src/Form.js 1/2

```
45     cancelHandler() {
46       this.props.history.push('/');
47     }
48
49     render() {
50       return <div>
51         <TextInput id={"nom"} label={"Nom"} value={this.state.nom} onChange={this.changeHandler} />
52         <TextInput id={"prenom"} label={"Prénom"} value={this.state.prenom} onChange={this.changeHandler} />
53         <TextInput id={"bureau"} label={"Bureau"} value={this.state.bureau} onChange={this.changeHandler} />
54         <button className="btn btn-primary" onClick={this.saveHandler}>Enregistrer</button>
55         <button className="btn btn-secondary" onClick={this.cancelHandler}>Retour</button>
56         <pre>{JSON.stringify(this.state)}</pre>
57       </div>
58     }
```

# BOUTON ANNULER

- > Penser à faire le **bind** dans le constructeur.

/src/Form.js 2/2

```
1 import React, { Component } from 'react';
2 import ProfService from './ProfService';
3
4 class Form extends Component {
5   constructor(props) {
6     super(props);
7     this.state = { nom : "", prenom : "", bureau : '' , motdepasse : ''}
8
9     this.changeHandler = this.changeHandler.bind(this);
10
11    this.saveHandler = this.saveHandler.bind(this);
12    this.cancelHandler = this.cancelHandler.bind(this);
13 }
```

# CONCLUSION

- > Voilà qui clôture la présentation à la fois de React ainsi que le panorama depuis le back jusqu'au front en ayant mis en place toutes les étapes et vu des technologies qui aident à résoudre les problèmes.
  
- > Concernant React, il y aurait évidemment matière à aller beaucoup plus loin.
- > Cette présentation trace les grandes lignes pour démarrer et permet de se rendre compte de ses apports.
- > L'approche de construire une interface à l'aide de composants est un élément important.
  
- > S'il devait y avoir des prochains chapitres, ce serait les Hooks et un gestionnaire d'état comme Redux.