



**formation.tech**

# Formation React

Romain Bohdanowicz

Twitter : @bioub - Github : <https://github.com/bioub>

<http://formation.tech/>

# Présentations

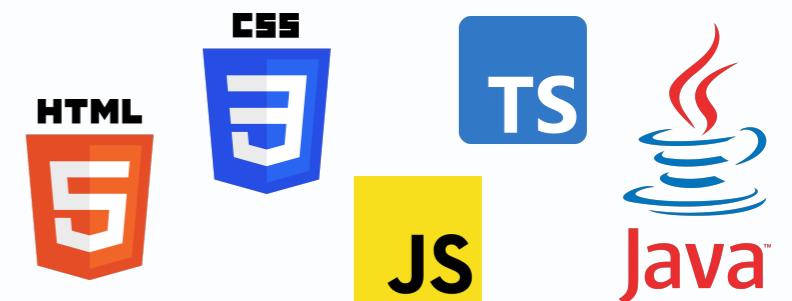


- Romain Bohdanowicz  
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle



- Expérience  
Formateur/Développeur Freelance depuis 2006  
Près de 2000 jours de formation animées

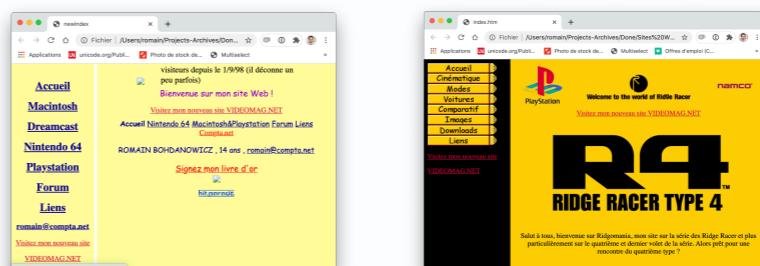
- Langages  
Expert : HTML / CSS / JavaScript / TypeScript / PHP / Java  
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch



- Certifications  
PHP / Zend Framework / Node.js



- A propos  
Premier site web à 12 ans (HTML/JS/PHP)  
Triathlète du dimanche





**formation.tech**

# Introduction

# Introduction



- React est une bibliothèque de création de composants capables de se rafraîchir à chaque changement d'état
- Crée par un employé Facebook en 2011
- Rendue Open-Source en 2013 (Licence BSD avec Clause)
- Licence MIT depuis novembre 2017
- Qui utilise React ?
  - Les produits de Facebook : Facebook, Instagram, WhatsApp...
  - AirBnb
  - Dropbox
  - Yahoo! Mail
  - formation.tech

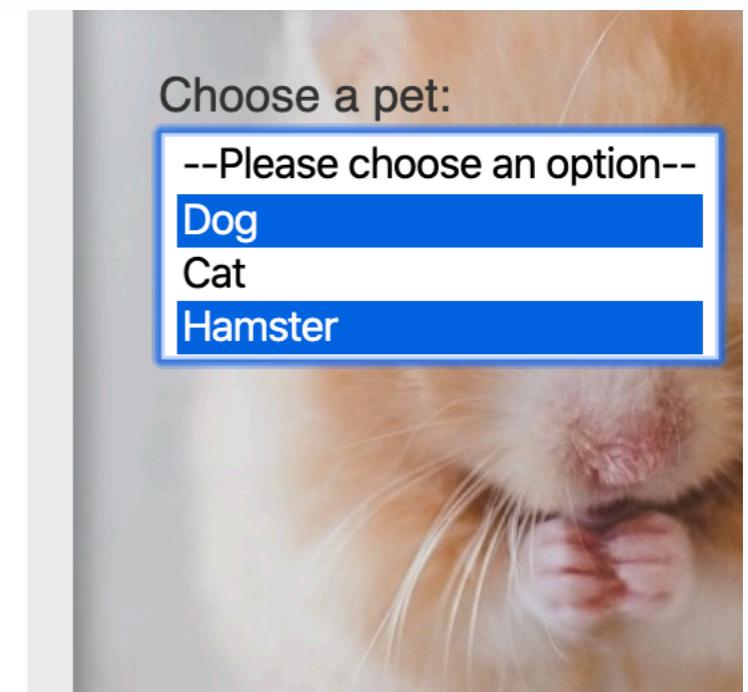
# Introduction - Qu'est-ce qu'un composant ?



- › Un composant est un moyen simple et isolé de regrouper :
  - Du HTML pour la structure de données
  - Du CSS pour la mise en forme
  - Du JavaScript pour le comportement
- › Exemple : la balise select

<https://developer.mozilla.org/fr/docs/Web/HTML/Element/select>

```
<label for="pet-select">Choose a pet:</label>  
  
<select name="pets" id="pet-select" multiple>  
  <option value="">--Please choose an option--</option>  
  <option value="dog">Dog</option>  
  <option value="cat">Cat</option>  
  <option value="hamster">Hamster</option>  
  <option value="parrot">Parrot</option>  
  <option value="spider">Spider</option>  
  <option value="goldfish">Goldfish</option>  
</select>
```





# Introduction - Qu'est-ce qu'un composant ?

- Le composant Select de react-select

<https://react-select.com/>

```
import React from 'react';

import Select from 'react-select';
import { colourOptions } from '../data';

export default () => (
  <Select
    defaultValue={[colourOptions[2], colourOptions[3]]}
    isMulti
    name="colors"
    options={colourOptions}
    className="basic-multi-select"
    classNamePrefix="select"
  />
);
```





# Introduction - Ecosystème

- A la différence de Google qui maintient tout un écosystème de bibliothèques dans Angular (pour gérer les requêtes HTTP, les formulaires, les pages ou les animations), React ne propose qu'un nombre limité de bibliothèques :
  - react  
Bibliothèque permettant la création de composants et la communication
  - react-dom  
Permet de faire le rendu d'un composant dans le navigateur via l'API DOM
  - react-dom/server  
Permet de faire le rendu d'un composant dans Node.js
  - react-native  
Permet de faire le rendu d'un composant dans une application native iOS ou Android
  - create-react-app  
Programme en CLI qui permet la création d'un squelette d'application React

# Introduction - Documentation



- React  
<https://reactjs.org/> (existe également en Français : <https://fr.reactjs.org/>)
- Tutoriel Officiel  
<https://reactjs.org/tutorial/tutorial.html>
- Blog  
<https://reactjs.org/blog/>
- Create React App  
<https://create-react-app.dev/>

# Introduction - Comment utiliser React ?



- Pour installer React on peut soit ajouter les balises script directement  
<https://reactjs.org/docs/add-react-to-a-website.html>
- Soit utiliser le programme create-react-app qui va créer la structure d'une application permettant l'utilisation de technologies comme JSX, TypeScript ou SASS
- Pour utiliser React simplement côté serveur : Next.js  
<https://nextjs.org/>
- Si on veut créer un site web statique : Gatsby  
<https://www.gatsbyjs.com/>
- Pour aller plus loin :
  - Neutrino : <https://neutrinojs.org/>
  - nwb : <https://github.com/insin/nwb>
  - Nx : <https://nx.dev/react>



**formation.tech**

# Create React App

# Create React App - Introduction



- Create React App est un programme CLI pour Node.js qui permet la création d'un nouveau squelette d'application React
- Pas besoin de l'installer globalement avec npm ou Yarn, il ne sert que pour créer le projet
- Création du projet
  - `npx create-react-app [chemin-projet] [options]`
  - `npm init react-app [chemin-projet] [options]`
  - `yarn create react-app [chemin-projet] [options]`

# Create React App - Introduction



- Options :
  - --use-npm ou --use-pnp sinon l'install se fait via Yarn quand il est installé
  - --template pour utiliser un autre squelette
  - --scripts-version pour utiliser un fork ou une autre version de react-scripts
- Trouver des templates  
[https://www.npmjs.com/search?q=cra-template-\\*](https://www.npmjs.com/search?q=cra-template-*)
- Exemple  
`npx create-react-app hello-react --use-npm --template @formation.tech/cra-template-basic`

# Create React App - Squelette cra-template



- Les deux templates officiel sont cra-template et cra-template-typescript
  - <https://github.com/facebook/create-react-app/tree/master/packages/cra-template>
  - <https://github.com/facebook/create-react-app/tree/master/packages/cra-template-typescript>
- Exemple
  - npx create-react-app hello-react

```
hello-react
├── README.md
├── node_modules
├── package.json
└── public
    ├── favicon.ico
    ├── index.html
    ├── logo192.png
    ├── logo512.png
    ├── manifest.json
    └── robots.txt
└── src
    ├── App.css
    ├── App.js
    ├── App.test.js
    ├── index.css
    ├── index.js
    ├── logo.svg
    ├── serviceWorker.js
    └── setupTests.js
yarn.lock
```



# Create React App - Squelette cra-template

- › Le package.json

```
{  
  "name": "hello-react",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "@testing-library/jest-dom": "^4.2.4",  
    "@testing-library/react": "^9.3.2",  
    "@testing-library/user-event": "^7.1.2",  
    "react": "^16.12.0",  
    "react-dom": "^16.12.0",  
    "react-scripts": "3.3.1"  
  },  
  "scripts": {  
    "start": "react-scripts start",  
    "build": "react-scripts build",  
    "test": "react-scripts test",  
    "eject": "react-scripts eject"  
  },  
  "eslintConfig": {  
    "extends": "react-app"  
  },  
  "browserslist": {  
    "production": [  
      ">0.2%",  
      "not dead",  
      "not op_mini all"  
    ],  
    "development": [  
      "last 1 chrome version",  
      "last 1 firefox version",  
      "last 1 safari version"  
    ]  
  }  
}
```



# Create React App - Avantages

- Create React App vous aide à :
  - Builder l'application
  - Lancer les tests avec Jest
  - Intégrer des fichiers CSS, des images, des assets...
  - Utiliser des fichiers de configuration



**formation.tech**

# React

# React - Premier composant



- React.createElement permet de créer un élément du DOM (un peu comme document.createElement)
  - 3 paramètres : nom de l'élément ou composant, propriétés, contenu
  - un composant est une fonction qui factorise un React.createElement
  - les composants commencent toujours par une majuscule
- ReactDOM.render permet le rendu de cet élément dans le navigateur via l'API DOM (voir aussi : react-native, react-dom/server, Ink...)

```
import React from 'react';

function App() {
  return React.createElement('div', { className: 'App' }, 'Hello React');
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(React.createElement(App), document.getElementById('root'));
```

# React - Premier composant



- › Voyons maintenant comment faire le rendu de plusieurs éléments :

```
import React from 'react';

function App() {
  // retourne <div class="App">Hello <span>React</span> !</div>
  return React.createElement(
    'div',
    { className: 'App' },
    'Hello ',
    React.createElement('span', {}, 'React'),
    ' !',
  );
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(React.createElement(App), document.getElementById('root'));
```

# React - Premier composant



- › Sans React, cela aurait été :

```
function app() {  
  const divEl = document.createElement('div');  
  divEl.className = 'App';  
  
  const helloTxt = document.createTextNode('Hello ');  
  divEl.appendChild(helloTxt);  
  
  const spanEl = document.createElement('div');  
  spanEl.innerText = 'React';  
  divEl.appendChild(spanEl);  
  
  const exclTxt = document.createTextNode(' ! ');  
  divEl.appendChild(exclTxt);  
  
  return divEl;  
}  
  
export default app;
```

```
import app from './app';  
  
const rootEl = document.getElementById('root');  
rootEl.appendChild(app());
```

- › React est donc moins verbeux et supporte d'autres plateformes que le navigateur !

# React - JSX



- › Facebook a également inventé une syntaxe appelée JSX (JavaScript XML) qui allège encore plus la création de composant :

```
import React from 'react';

function App() {
  return (
    <div className="App">
      Hello <span>React</span> !
    </div>
  );
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

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

- › Documentation  
<https://facebook.github.io/jsx/>
- › Langage proche du HTML nécessitant une compilation (avec Babel par exemple et son plugin babel-plugin-transform-react-jsx)

# React - JSX



- Jusqu'à React 16 il fallait importer React (car les balises JSX se transforment en React.createElement)
- Les parenthèses après le return sont nécessaires pour pouvoir indenter (sinon le retour à la ligne sera vu comme la fin du return)
- Attention à ne pas confondre propriétés du DOM et attributs HTML (class vs className)
- Les balises vides se terminent par un slash comme en XML (<App />)

```
import React from 'react';

function App() {
  return (
    <div className="App">
      Hello <span>React</span> !
    </div>
  );
}

export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';

import App from './App';

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

# React - JSX



- › Pour concaténer une expression on utilise les accolades {}

```
function App() {  
  const name = 'React';  
  return (  
    <div className="App">  
      Hello {name} !  
    </div>  
  );  
}  
  
export default App;
```

- › React ne sait pas afficher des objets

```
function App() {  
  const obj = {  
    name: 'React',  
  };  
  return (  
    <div className="App">  
      Hello {obj} ! /* Error: Objects are not valid as a React child */  
    </div>  
  );  
}  
  
export default App;
```

# React - JSX



- Les commentaires sont des commentaire JS entre accolades :

```
import React from 'react';

function App() {
  return (
    <div className="App">
      {/* Hello <span>React</span> ! */}
    </div>
  );
}

export default App;
```

- Privilégier le raccourci de VSCode.

# React - Props



- Pour passer des paramètres à un composant on utilise les props :
  - Même syntaxe que les propriétés du DOM
  - Les props sont regroupés dans le paramètre d'entrée du composant
  - A l'appel, une chaîne de caractères utilise les guillemets, les autres types les accolades

```
function Hello(props) {  
  return (  
    <div className="Hello">  
      Hello my name is {props.name}, I'm {props.age} !!!  
    </div>  
  );  
}  
  
export default Hello;
```

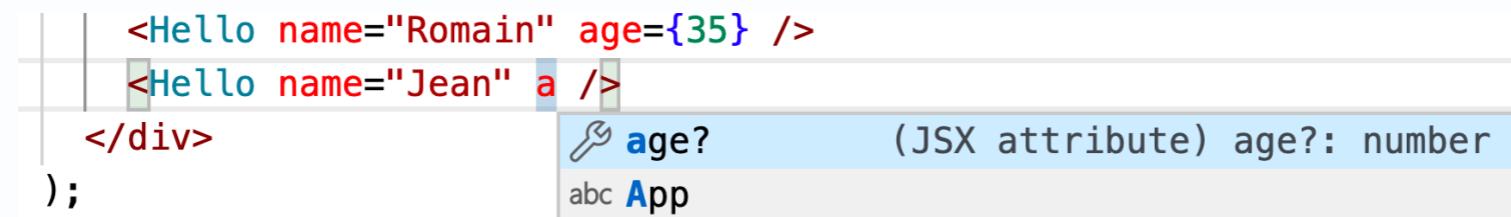
```
import Hello from './Hello';  
  
function App() {  
  return (  
    <div className="App">  
      <Hello name="Romain" age={35} />  
      <Hello name="Jean" age={60} />  
    </div>  
  );  
}  
  
export default App;
```

# React - Props



- Il est recommandé de déstructurer l'objet props :
  - les lignes sont plus courtes
  - on peut passer des valeurs par défaut facilement
  - VSCode affiche des suggestions dans l'IntelliSense
  - Voir aussi prop-types, Flow et TypeScript

```
function Hello({ name = '', age = 0 }) {  
  return (  
    <div className="Hello">  
      Hello my name is {name}, I'm {age} !!!  
    </div>  
  );  
}  
  
export default Hello;
```



# React - State



- Contrairement à d'autres bibliothèques similaires, React ne rafraîchit par le DOM lorsqu'une variable change de valeur

```
function Clock() {
  let now = new Date();
  setInterval(() => {
    now = new Date(); // sans effet sur le DOM
  }, 1000);

  return <div className="Clock">{now.toLocaleTimeString()}</div>;
}

export default Clock;
```

```
import Clock from './Clock';

function App() {
  return (
    <div className="App">
      <Clock />
    </div>
  );
}

export default App;
```

# React - State



- Pour que le composant se rafraîchisse sur une valeur interne, on utilise le state :
  - nécessite l'utilisation d'une classe (ou des hooks de React 16.8)
  - l'objet state doit être initialisé (null sinon)
  - la méthode setState permet de mettre à jour le state et donc le DOM

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

class Clock extends Component {
  constructor() {
    super();
    this.state = {
      now: new Date(),
    };
    setInterval(() => {
      this.setState({
        now: new Date(),
      });
    }, 1000);
  }
  render() {
    const { now } = this.state;

    return <div className="Clock">{now.toLocaleTimeString()}</div>;
  }
}
export default Clock;
```

# React - State



- Il n'est pas nécessaire de repasser tout le state à la méthode setState
- Son fonctionnement est similaire à la fonction Object.assign (pas récursif donc)

```
import { format as formatDate } from 'date-fns';
import { Component } from 'react';

class Clock extends Component {
  constructor() {
    super();
    this.state = {
      format: 'HH:mm:ss',
      now: new Date(),
    };
    setInterval(() => {
      this.setState({
        now: new Date(),
      });
    }, 1000);
  }
  render() {
    const { format, now } = this.state;

    return <div className="Clock">{formatDate(now, format)}</div>;
  }
}

export default Clock;
```



# React - State + props

- Un "class component" peut accéder aux props dans le constructeur

```
import { format as formatDate } from 'date-fns';
import { Component } from 'react';

class Clock extends Component {
  constructor({ format = 'HH:mm:ss' }) {
    super();
    this.state = {
      nowStr: formatDate(new Date(), format),
    };
    setInterval(() => {
      this.setState({
        nowStr: formatDate(new Date(), this.props.format),
      });
    }, 1000);
  }
  render() {
    const { nowStr } = this.state;

    return <div className="Clock">{nowStr}</div>;
  }
}

export default Clock;
```

# React - State + props



- Dans une autre méthode via `this.props`

```
import { format as formatDate } from 'date-fns';
import { Component } from 'react';

class Clock extends Component {
  constructor() {
    super();
    this.state = {
      now: new Date(),
    };
    setInterval(() => {
      this.setState({
        now: new Date(),
      });
    }, 1000);
  }
  render() {
    const { now } = this.state;
    const { format = 'HH:mm:ss' } = this.props;

    return <div className="Clock">{formatDate(now, format)}</div>;
  }
}

export default Clock;
```

# React - State + props



- Evitez de créer le state à partir des props

```
import { format as formatDate } from 'date-fns';
import { Component } from 'react';

class Clock extends Component {
  constructor({ now = new Date() }) {
    super();
    this.state = {
      now,
    };
    setInterval(() => {
      this.setState({
        // on met à jour la date ? on garde celle des props ?
      });
    }, 1000);
  }
  render() {
    const { now } = this.state;

    return <div className="Clock">{now.toLocaleTimeString()}</div>;
  }
}

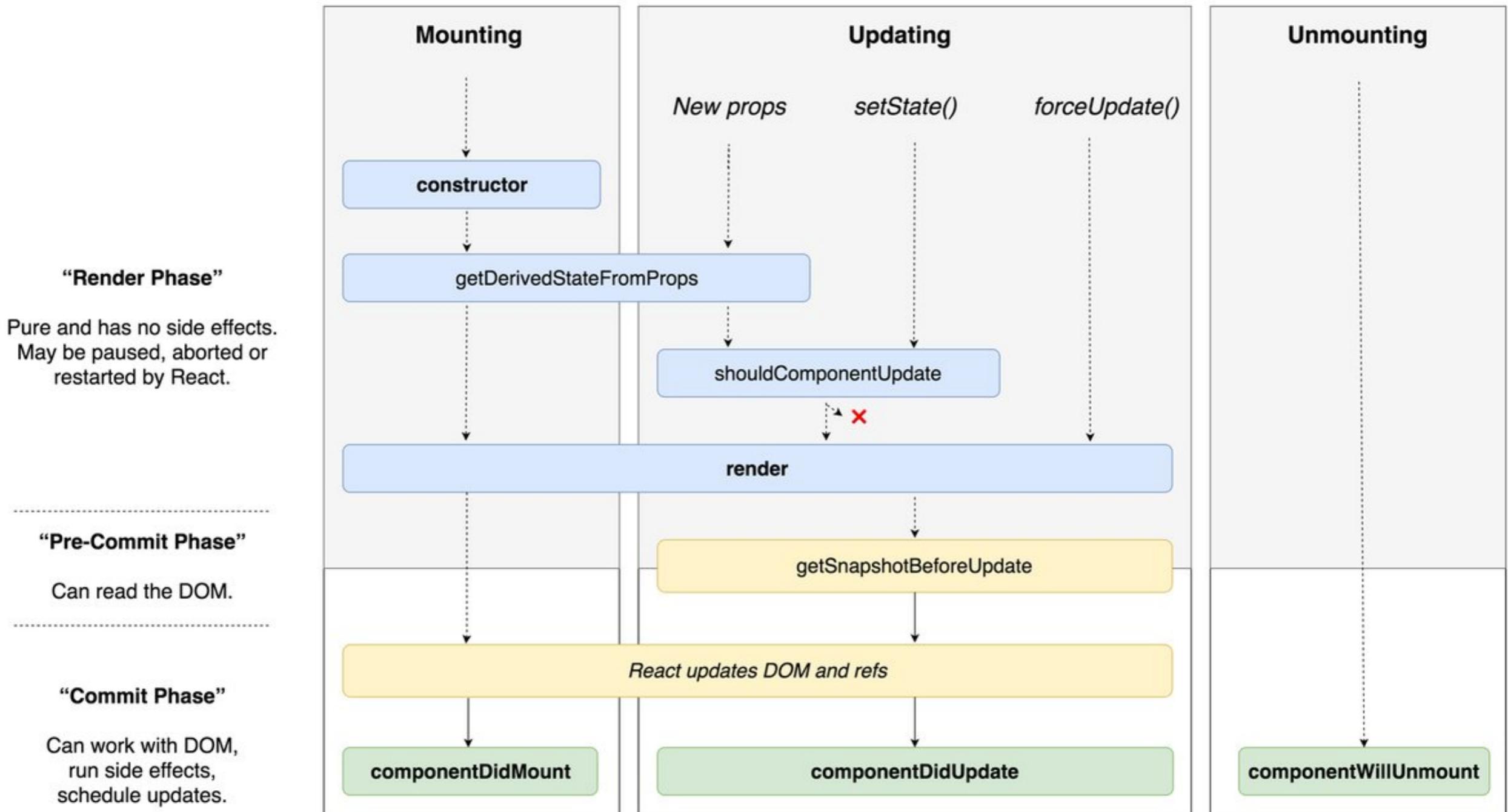
export default Clock;
```



# React - Cycle de vie

- Dans un "class component", certaines méthodes sont appelées automatiquement
- Montage (le composant apparaît dans le DOM pour la première fois :
  - constructor()
  - getDerivedStateFromProps()
  - render()
  - componentDidMount()
- Mise à jour (nouvelle valeur de props, appel à setState ou forceUpdate)
  - getDerivedStateFromProps()
  - shouldComponentUpdate()
  - render()
  - getSnapshotBeforeUpdate()
  - componentDidUpdate()
- Démontage (le composant disparaît du DOM)
  - componentWillUnmount()

# React - Cycle de vie



# React - Cycle de vie



- `constructor()`
  - Appelée côté client et serveur
  - `constructor` sert à initialiser state et props

# React - Cycle de vie



- `componentDidMount()`
  - Le rendu initial du composant a été effectué
  - Il est possible de manipuler le DOM
  - N'existe que côté client
  - Le bon endroit pour charger un plugin jQuery ou tout ce qui ne s'exécute qu'une seul fois et qui a besoin d'accéder au DOM
  - Démarrer des requêtes AJAX, des timers...



# React - Cycle de vie

- componentDidMount()

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

class Clock extends Component {
  constructor() {
    super();
    this.state = {
      now: new Date(),
    };
  }
  componentDidMount() {
    setInterval(() => {
      this.setState({
        now: new Date(),
      });
    }, 1000);
  }
  render() {
    const { now } = this.state;

    return <div className="Clock">{now.toLocaleTimeString()}</div>;
  }
}

export default Clock;
```

# React - Cycle de vie



- `shouldComponentUpdate()`
  - Permet d'empêcher un `render()`, doit répondre true ou false
  - Utile pour optimiser une application, ne pas faire de rendu si les props ou le state on été modifié d'une manière qui ne nécessite pas un nouveau rendu (voir aussi `PureComponent`)
- `getDerivedStateFromProps()`
  - En général inutilisé : <https://fr.reactjs.org/docs/react-component.html#static-getderivedstatefromprops>
- `getSnapshotBeforeUpdate()`
  - En général inutilisé : <https://fr.reactjs.org/docs/react-component.html#getSnapshotBeforeUpdate>



# React - Cycle de vie

- `componentDidUpdate()`
  - Juste après un rendu autre que initial
  - On a accès au DOM
  - Le bon endroit pour un update d'un plugin jQuery (Chosen, Select2...)
- `componentWillUnmount()`
  - Le composant va être supprimer
  - Permet de supprimer des listeners, libérer la mémoire, appeler clearInterval/Timeout, sinon l'objet associé au composant ne sera jamais détruit (si ref interne dans un callback)

# React - Cycle de vie



- componentDidUpdate() / componentWillUnmount()

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

class Clock extends Component {
  constructor() {
    super();
    this.state = {
      now: new Date(),
    };
  }
  componentDidUpdate(prevProps, prevState, snapshot) {
    if (this.props.delay !== prevProps.delay) {
      this._interval = setInterval(() => {
        this.setState({
          now: new Date(),
        });
      }, this.props.delay);
    }
  }
  componentWillUnmount() {
    clearInterval(this._interval);
  }
  render() {
    const { now } = this.state;

    return <div className="Clock">{now.toLocaleTimeString()}</div>;
  }
}

export default Clock;
```

# React - Cycle de vie



- D'autres méthodes ont existé et peuvent être présente dans du code legacy
  - componentWillMount
  - componentWillReceiveProps
  - componentWillUpdate
  - UNSAFE\_componentWillMount
  - UNSAFE\_componentWillReceiveProps
  - UNSAFE\_componentWillUpdate
- Les versions UNSAFE\_ évitaient un warning
- <https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html>

# React - Evénements



- On écoute les événements du DOM via les props spéciales on\*

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

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState({
      count: this.state.count + 1,
    });
  }
  render() {
    const { count } = this.state;

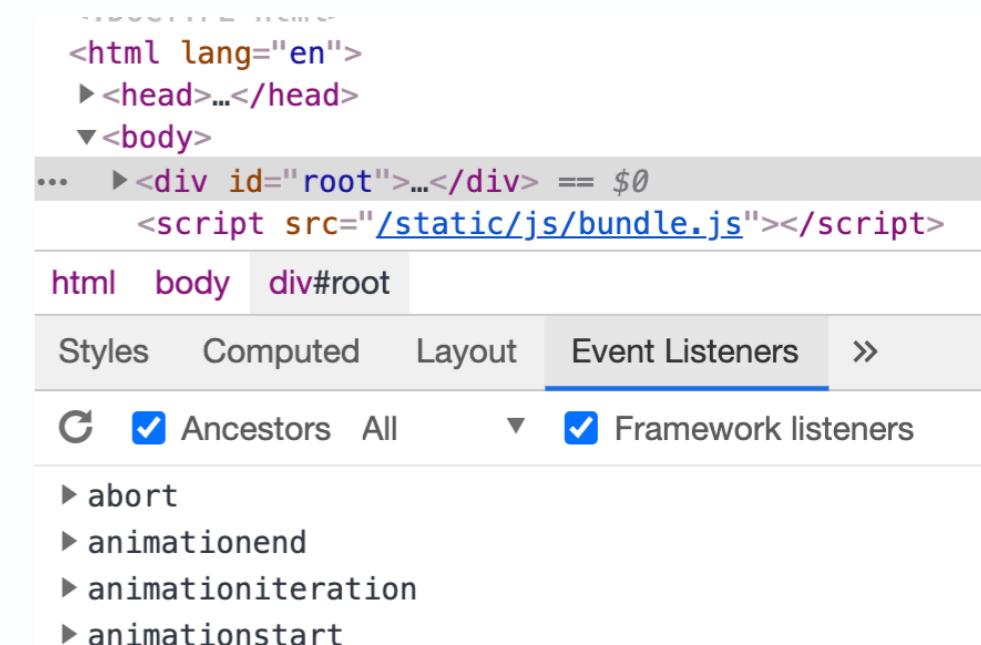
    return (
      <button className="Counter" onClick={this.handleClick}>
        {count}
      </button>
    );
  }
}

export default Counter;
```

# React - Evénements



- › Si on avait dû utiliser addEventListener il aura fallu penser au removeEventListener dans componentDidMount
- › Certains événements sont réinterprétés (onChange)
- › Les objets Event natif sont encapsulés dans des SyntheticEvent React pour assurer une meilleure compatibilité entre les navigateurs :  
<https://fr.reactjs.org/docs/events.html>
- › Les props on\* écoutent les événements natifs :
  - Au niveau de l'objet document (avant React 17)
  - Au niveau de la balise racine (React 17+)



The screenshot shows the Chrome DevTools Elements tab with the DOM tree. The root element is a `<div id="root">`. The `Event Listeners` tab is selected, showing four event types: `abort`, `animationend`, `animationiteration`, and `animationstart`. The `Framework listeners` checkbox is checked.

```
<html lang="en">
  <head>...</head>
  <body>
    <div id="root">...</div> == $0
    <script src="/static/js/bundle.js"></script>

```

# React - Evénements



- › L'objet SyntheticEvent contient le même API que Event
- › Pour accéder à l'objet event natif : event.nativeEvent

```
import { Component, SyntheticEvent } from "react";

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0,
    };
    // pour accéder à this dans handleClick
    this.handleClick = this.handleClick.bind(this);
  }
  /** @param {SyntheticEvent<HTMLButtonElement, MouseEvent>} event */
  handleClick(event) {
    this.setState({
      count: Number(event.currentTarget.innerText) + 1,
    });
  }
  render() {
    const { count } = this.state;
    return (
      <button className="Counter" onClick={this.handleClick}>
        {count}
      </button>
    );
  }
}
```



# React - Evénements

- › On doit "binder" le callback pour que this fasse référence au composant React
- › Par défaut this === undefined sinon

```
constructor(props) {  
  super(props);  
  this.state = {  
    count: 0,  
  };  
  // pour accéder à this dans handleClick  
  this.handleClick = this.handleClick.bind(this);  
}
```

# React - Evénements



- › On peut également "binder" dans le render
- › Attention une nouvelle fonction sera créée à chaque render

```
render() {
  const { count } = this.state;

  return (
    <button className="Counter" onClick={this.handleClick.bind(this)}>
      {count}
    </button>
  );
}
```



# React - Evénements

- › On peut utiliser une fonction fléchée dans le callback
- › Attention une nouvelle fonction sera créée à chaque render

```
render() {
  const { count } = this.state;

  return (
    <button className="Counter" onClick={() => this.handleClick()}>
      {count}
    </button>
  );
}
```

# React - Evénements



- On peut également déclarer directement la propriété avec les class properties de ESNext
- Attention cette syntaxe n'est pas encore normée

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

class Counter extends Component {
  state = {
    count: 0,
  };
  handleClick = () => {
    this.setState({
      count: this.state.count + 1,
    });
  }
  render() {
    const { count } = this.state;

    return (
      <button className="Counter" onClick={this.handleClick}>
        {count}
      </button>
    );
  }
}

export default Counter;
```

# React - Evénements



- Comment passer un paramètre autre que l'objet event ?
- Avec une fonction fléchée
- Attention une nouvelle fonction sera créée à chaque render

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

class Counter extends Component {
  state = {
    count: 0,
  };
  handleClick(count) {
    this.setState({
      count: count + 1,
    });
  }
  render() {
    const { count } = this.state;

    return (
      <button className="Counter" onClick={() => this.handleClick(count)}>
        {count}
      </button>
    );
  }
}

export default Counter;
```

# React - Evénements



- Comment passer un paramètre autre que l'objet event ?
- Avec une fonction fléchée
- Attention une nouvelle fonction sera créée à chaque render

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

class Counter extends Component {
  state = {
    count: 0,
  };
  handleClick(count) {
    this.setState({
      count: count + 1,
    });
  }
  render() {
    const { count } = this.state;

    return (
      <button className="Counter" onClick={() => this.handleClick(count)}>
        {count}
      </button>
    );
  }
}

export default Counter;
```

# React - Evénements



- En passant par les data-attributs :

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

class Counter extends Component {
  state = {
    count: 0,
  };
  handleClick = (event) => {
    this.setState({
      count: Number(event.target.dataset.count) + 1,
    });
  }
  render() {
    const { count } = this.state;

    return (
      <button className="Counter" data-count={count} onClick={this.handleClick}>
        {count}
      </button>
    );
  }
}

export default Counter;
```

# React - Rendu conditionnel



- Le rendu conditionnel peut se faire en affectant du JSX ou undefined à une variable

```
render() {
  let clock;
  let button;

  const { showClock } = this.state;

  if (showClock) {
    clock = <Clock />;
  }

  if (!showClock) {
    button = <button onClick={this.handleClick}>On</button>;
  } else {
    button = <button onClick={this.handleClick}>Off</button>;
  }

  return (
    <div className="App">
      {button} {clock}
    </div>
  );
}
```

# React - Rendu conditionnel



- Selon les cas il sera plus ou moins lisible d'utiliser les opérateurs && et ternaire directement dans le JSX

```
render() {
  const { showClock } = this.state;

  return (
    <div className="App">
      {!showClock ? (
        <button onClick={this.handleClick}>On</button>
      ) : (
        <button onClick={this.handleClick}>Off</button>
      )}
      {showClock && <Clock />}
    </div>
  );
}
```

# React - Rendu conditionnel



- On peut également retourner null dans un composant pour ne pas effectuer son rendu

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

class Clock extends Component {
  state = {
    now: new Date(),
  };
  render() {
    if (!this.props.show) {
      return null;
    }

    return <div className="Clock">{this.state.now.toLocaleTimeString()}</div>;
  }
}

export default Clock;
```

# React - Listes



- React peut faire le rendu d'un tableau de chaîne de caractères

```
function List() {  
  const letters = ['A', 'B', 'C'];  
  
  return (  
    <ul>  
      {letters}  
    </ul>  
  );  
}  
  
export default List;
```

```
▼<ul>  
  "A"  
  "B"  
  "C"  
</ul>
```

- Ou d'un tableau de JSX

```
function List() {  
  const letters = [<li key="A">A</li>, <li key="B">B</li>, <li key="C">C</li>];  
  
  return (  
    <ul>  
      {letters}  
    </ul>  
  );  
}  
  
export default List;
```



# React - Listes

- Avant la fonction map on peut transformer n'importe quel tableau en JSX simplement :

```
function List() {
  const letters = ['A', 'B', 'C'];
  const lettersJSX = letters.map((l) => <li key={l}>{l}</li>);

  return (
    <ul>
      {lettersJSX}
    </ul>
  );
}

export default List;
```

- Ou directement dans le JSX

```
function List() {
  const letters = ['A', 'B', 'C'];

  return (
    <ul>
      {letters.map((l) => <li key={l}>{l}</li>)}
    </ul>
  );
}

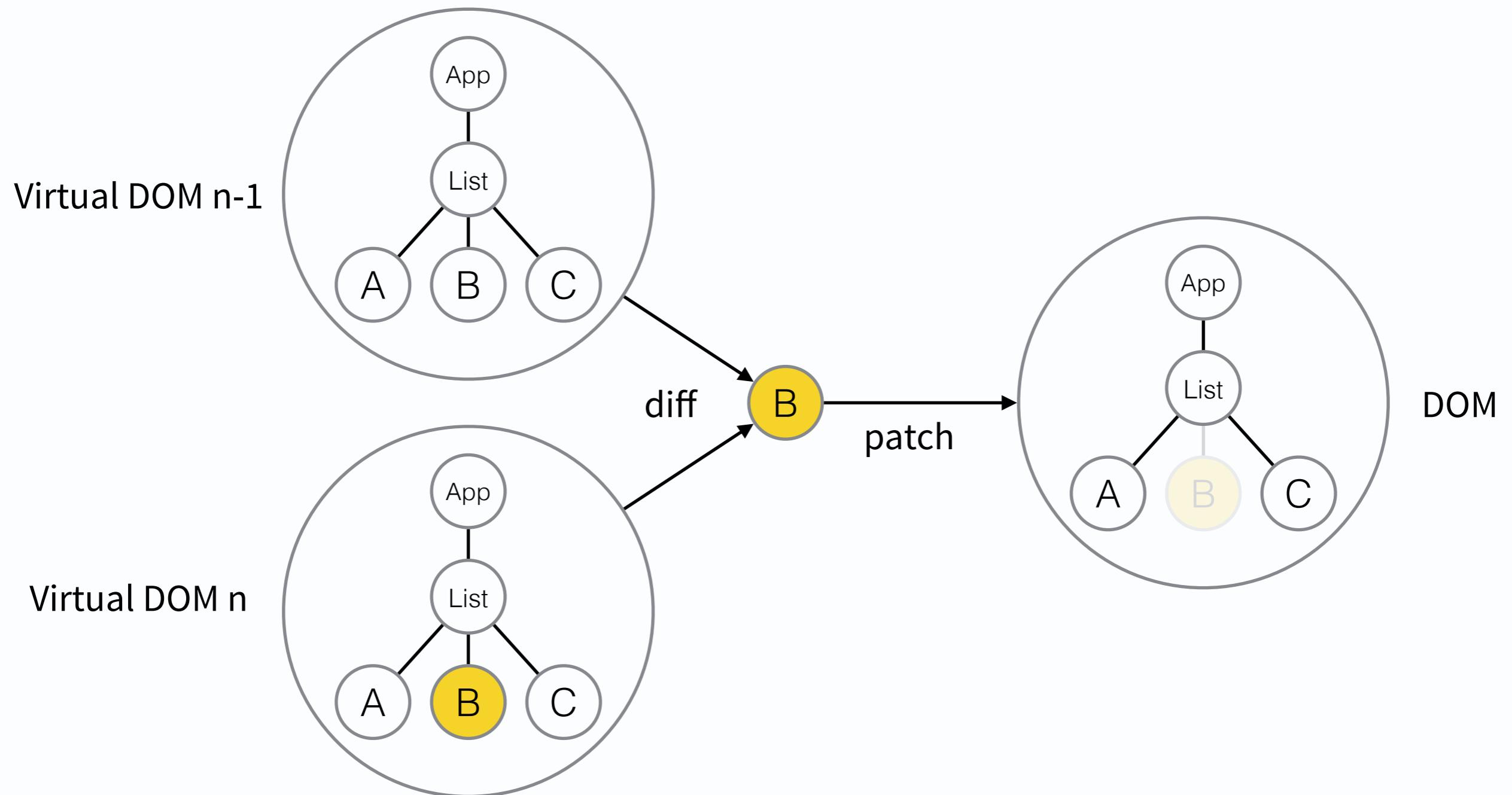
export default List;
```

# React - Clés



- Lorsqu'on utilise un tableau de JSX il est important de spécifier une clé avec la props key
- Elle permet à React de minimiser le nombre d'opérations à effectuer dans le DOM en cas d'update
- Elle doit être unique dans le tableau (pas besoin d'être unique dans toute l'application)
- Elle ne doit pas être modifiée d'un render à un autre
  - Eviter les valeurs générées dans le render
  - Eviter les indices des tableaux qui évoluent avec les tris, filtres, suppressions...

# React - Réconciliation





# React - Formulaires

```
class LoginForm extends Component {
  state = {
    login: '',
    password: '',
  };
  handleSubmit = (event) => {
    event.preventDefault();
    console.log(this.state);
  };
  handleChange = (event) => {
    this.setState({
      [event.target.name]: event.target.value,
    });
  };
  render() {
    return (
      <form className="LoginForm" onSubmit={this.handleSubmit}>
        <div>
          Login :{' '}
          <input type="text" name="login" onChange={this.handleChange} />
        </div>
        <div>
          Password :{' '}
          <input type="password" name="password" onChange={this.handleChange} />
        </div>
        <div>
          <button>Login</button>
        </div>
      </form>
    );
  }
}
```

# React - Imbrication de composants



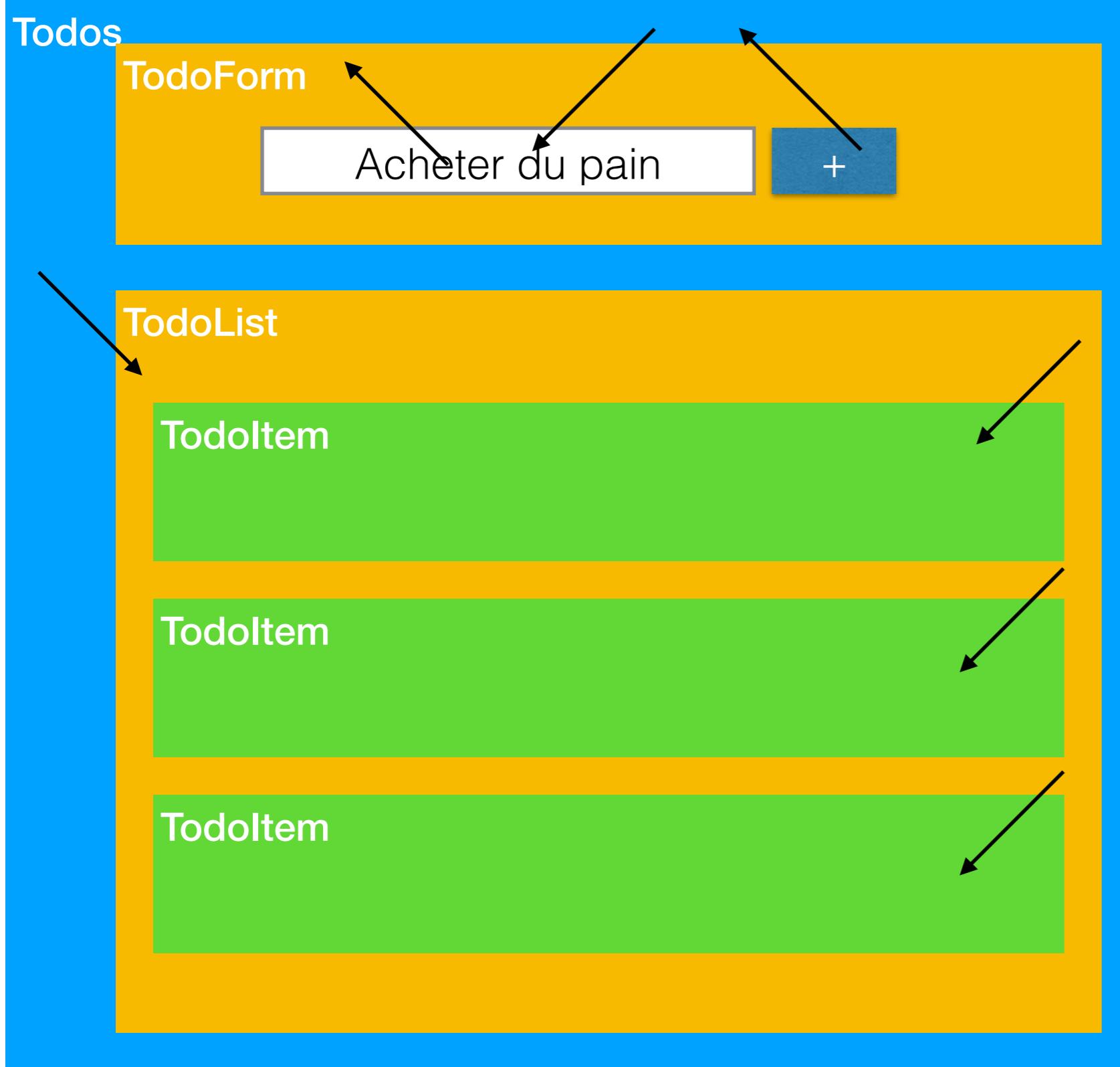
- Lorsque qu'un state doit être accessible par plusieurs composants, il faut le définir sur le plus proche ancêtre commun, les composants imbriqués y accéder au travers de props

```
class App extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }
  increment() {
    this.setState({ count: this.state.count + 1 });
  }
  decrement() {
    this.setState({ count: this.state.count - 1 });
  }
  render() {
    return <div className="App">
      <h1>{this.state.count}</h1>
      <CounterButton update={this.increment.bind(this)}>+</CounterButton>
      <CounterButton update={this.decrement.bind(this)}>-</CounterButton>
    </div>;
  }
}

class CounterButton extends React.Component {
  render() {
    return <button onClick={this.props.update}>
      {this.props.children}
    </button>;
  }
}
```



# React - Exercice Communication



- Créer 4 composants :
  - Todos -> Parent
  - TodoForm -> HelloWorld / CounterControlled
  - TodoList -> Hello avec une boucle
  - TodoItem -> Hello
- Dans le state de Todos créer une liste de todos (object[]) avec 3 clés : id, title, completed) et newTodo de type string
- Passer ce tableau à TodoList, qui créera autant de TodoItem qu'il y a d'élément dans le tableau
- TodoItem reçoit un élément et l'affiche
- TodoForm Controlled Component, reçoit 3 props : newTodo et 2 callback de Todos et lui passe la valeur saisie au submit du form
- 2 callbacks de TodoForm :
  - éditer la clé newTodo (onNewTodoChange event change)
  - ajoute l'élément au tableau (onAdd event submit)