

React

Développer avec ReactJS

Animé par Mazen Gharbi

Introduction

- ▷ Librairie créée en 2013 par Facebook, directement inspirée de la surcouche PHP XHP ;
 - › *extension de PHP pour permettre la syntaxe XML dans le but de créer des éléments HTML personnalisés et réutilisables*

PHP classique

```
if ($_POST['name']) {  
?>  
    Hello, <?=$_POST['name']?>.  
} else {  
?>  
    <form method="post">  
        What is your name?  
        <input type="text" name="name">  
        <input type="submit">  
    </form>  
}
```

Avec XHP

```
if ($_POST['name']) {  
    echo Hello, {$_POST['name']};  
} else {  
    echo  
    <form method="post">  
        What is your name?  
        <input type="text" name="name" />  
        <input type="submit" />  
    </form>;  
}
```

Introduction

- Implémente la même logique côté front avec Javascript
- Voici un exemple de code React :

Vue d'un programme React

```
{  
  this.state.error &&  
  <Text style={styles.error}>Une erreur est survenue</Text>  
}  
  
<Text style={{ fontSize: 17, marginTop: 10, color: '#C1C1C1', textAlign: 'center', padding: 10 }}>  
{  
  `veuillez renseigner les informations de votre client ci-dessous`  
}  
</Text>
```

React Native

Premiers pas

[Tester ce code](#)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8"/>
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
<div id="root"></div>
<script type="text/babel">
  ReactDOM.render (
    <h1>Hello, world!</h1>, document.getElementById('root')
  );
</script>
</body>
</html>
```

index.html

1
2
3

Code JSX

Comment ça marche

- ▷ Pour écrire une première page avec React, nous avons ajouté trois scripts: [React](#), [ReactDOM](#) et [Babel](#) ;
- ▷ La différence entre React et ReactDOM est que le premier gère le cœur de la librairie (composants, state, props) et le deuxième gère l'intégration avec les APIs DOM ;
- ▷ Cette séparation a été faite par les développeurs afin de permettre l'émergence de moteurs de rendu pour d'autres plateformes comme :
 - › [React Native](#) pour créer des applications mobiles ;
 - › [React Blessed](#) pour créer des interfaces sur le terminal ;
 - › [React VR](#) pour créer des sites web utilisant la VR ;

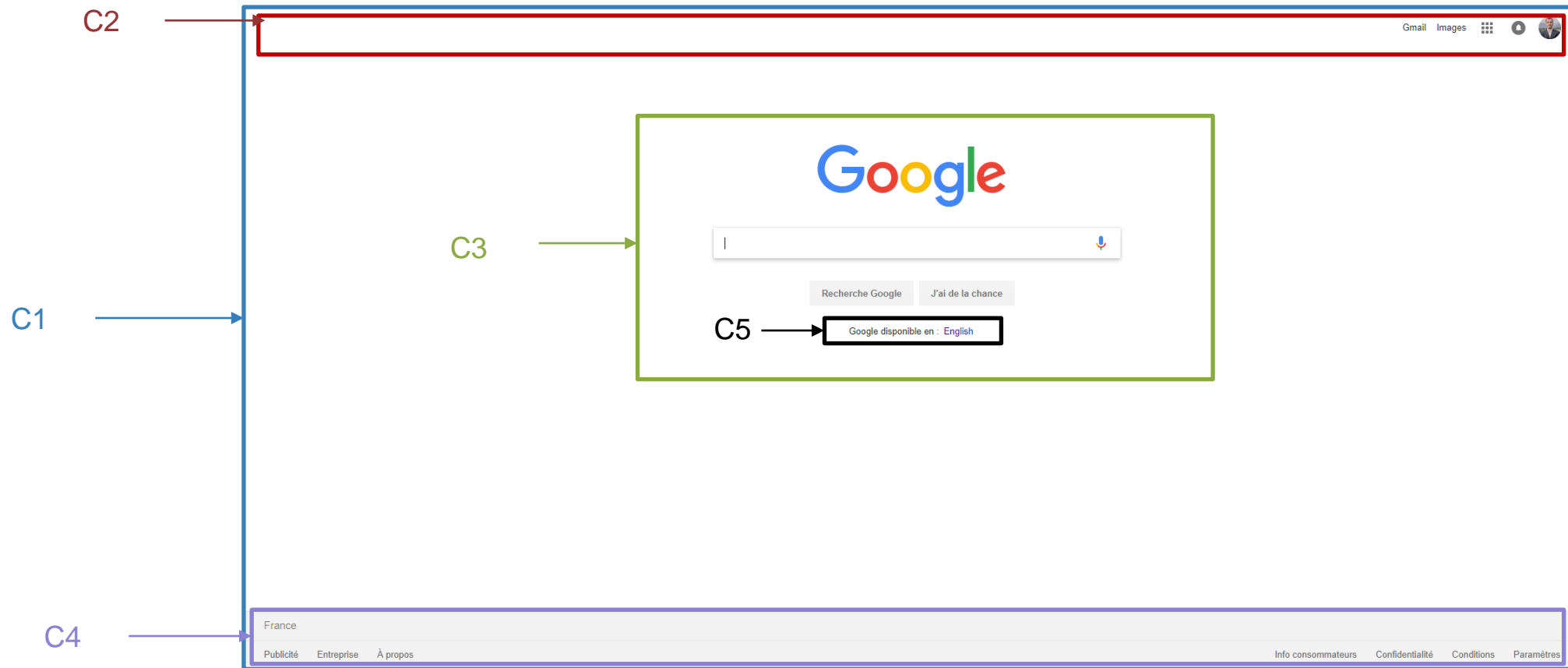


Les composants

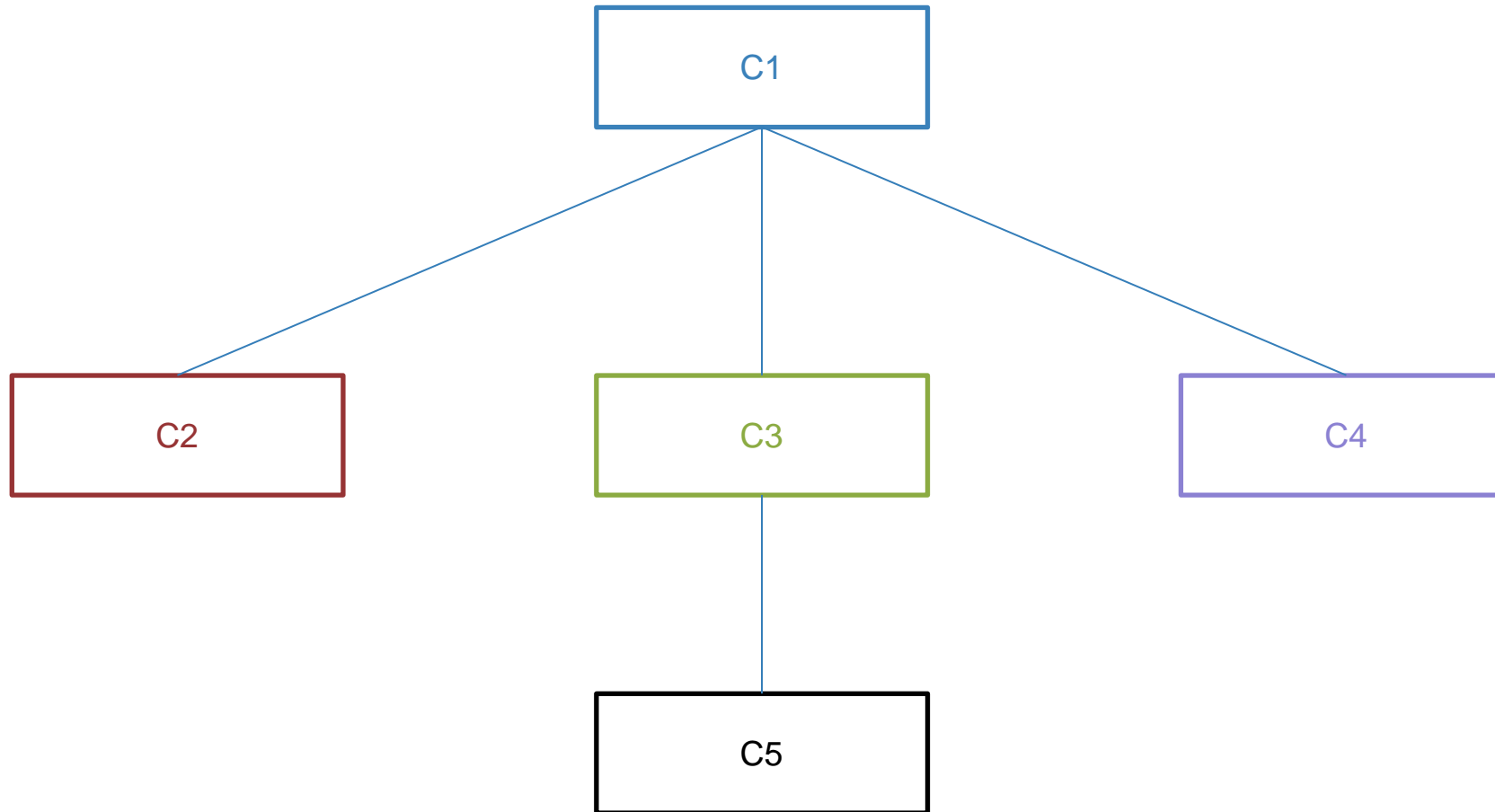
Pilier de la librairie

- ▷ Un composant contrôle une vue ou une partie d'une vue
- ▷ L'un des principaux concepts de React est de voir une application comme une **arborescence de composants**.
- ▷ Les composants permettent une meilleure décomposition de l'application, facilitent le **refactoring** et le **testing**.
- ▷ Chaque composant est **isolé** des autres composants. Il n'hérite pas implicitement des attributs des composants parents.

Séparation par composants



Séparation par composants



Composants React

- ▷ Un composant React peut être défini comme une classe ES6 qui étend la classe React :

```
export default class HelloScreen extends React.Component {
```

Importé au préalable



- ▷ Au minimum, un composant doit définir une méthode *render()* spécifiant le rendu du composant dans le DOM

- › *Cette fonction doit obligatoirement être présente*

- ▷ La méthode de *render()* renvoie les noeuds React, qui peuvent être définis à l'aide de la syntaxe JSX en tant que balises de type HTML

Functionnal Components React

- ▷ C'est la nouvelle bonne pratique !

```
export default function HelloScreen() { ...
```

- ▷ Pas besoin de render cette fois !
- ▷ Il suffit de renvoyer la vue directement

```
...  
return <div>Ma vue</div>;
```

Avec React, on peut...

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

App.js

```
class App extends Component {
```

```
  render() {
```

```
    return (
```

```
      <div className="App">
```

```
        <header className="App-header">
```

```
          <img src={logo} className="App-logo" alt="logo" />
```

```
          <h1 className="App-title">Welcome to React</h1>
```

```
        </header>
```

```
        <p className="App-intro">
```

```
          To get started, edit <code>src/App.js</code> and save to reload.
```

```
        </p>
```

```
      </div>
```

```
    );
```

```
  }
```

```
}
```

Fourni par React. Définit un composant de base !

JSX

Et voici l'équivalent en Javascript...

Développer avec ReactJS

```
React.createElement(  
  "div",  
  { className: "App" },  
  React.createElement(  
    "header",  
    { className: "App-header" },  
    React.createElement("img", { src: logo, className: "App-logo", alt: "logo" }),  
    React.createElement(  
      "h1",  
      { className: "App-title" },  
      "Welcome to React"  
    )  
  ),  
  React.createElement(  
    "p",  
    { className: "App-intro" },  
    "To get started, edit ",  
    React.createElement(  
      "code",  
      null,  
      "src/App.js"  
    ),  
    " and save to reload."  
  )  
);
```

C'est quand même vachement mieux en JSX

JSX

- ▷ Comme vous avez pu le constater, la vue est directement intégrée dans le modèle ;
- ▷ Les développeurs React ont poussé jusqu'au bout leur vision d'avoir la logique intimement reliée à un élément.
- ▷ Le contenu a été intégré dans le même fichier afin de n'en avoir qu'un seul par composant.

JSX

- ▷ Le navigateur ne comprend pas le JSX. Cette syntaxe étant invalide.
- ▷ Certains outils permettent de « transformer » le JSX en code valide
 - › BabelJS

Création du composant

```
import React from "react";

class HelloWorld extends React.Component {
  render() {
    return <h1>Hello, World!</h1>;
  }
}

export default HelloWorld;
```

HelloWorld.js

Appel du composant

```
import HelloWorld from './Hello';

class App extends Component {
  render() {
    return (
      <HelloWorld />
    );
  }
}
```

Fait le lien avec le
index.html

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

Avec React, on peut aussi...

▷ Intégrer aisément notre composant racine à notre page :

Fournit par react

main.js

```
import registerServiceWorker from './registerServiceWorker';
```

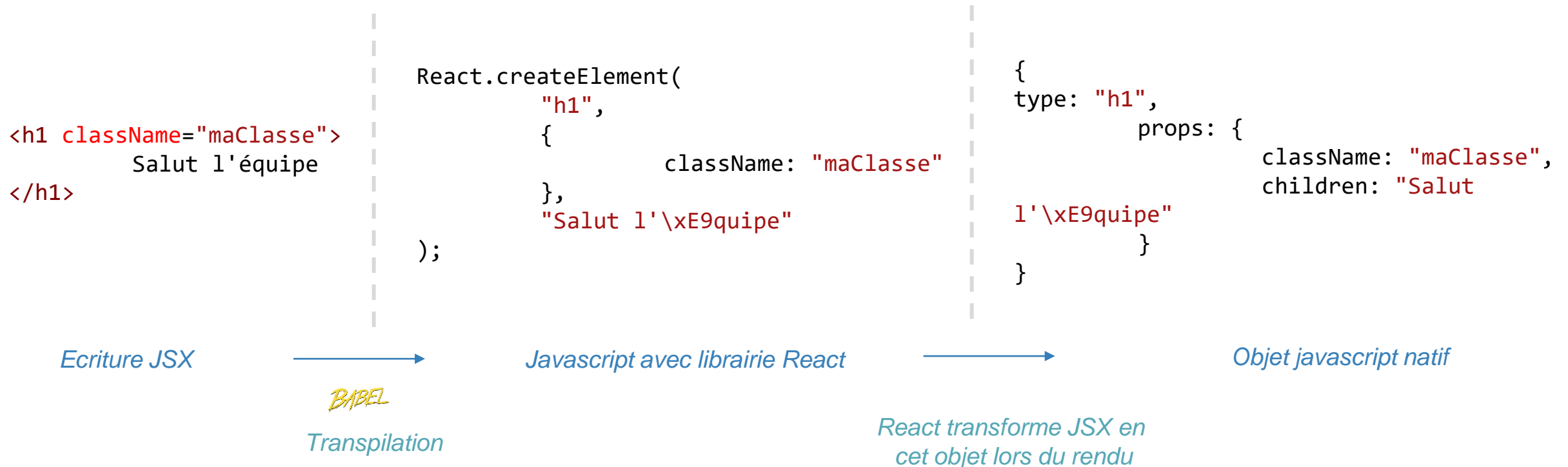
```
ReactDOM.render(<App />, document.getElementById('root'));  
registerServiceWorker();
```

Permet la création d'un ServiceWorker pour:

- Faire tourner l'appli offline (gestion cache)
- La mise en place d'une PWA par la suite

JSX

- Si on veut ajouter des attributs HTML, ils doivent avoir les mêmes noms qu'en créant des éléments en JS et écrits en camelCase :



JSX

▷ Si on veut écrire du JS dans notre composant, il faut l'entourer d'accolades :

JSX :

```
<h1 className="maClasse">  
  {"SaLUt l'éQUIPE !!".toLowerCase()}  
  {2 * 4}  
</h1>
```

HTML :

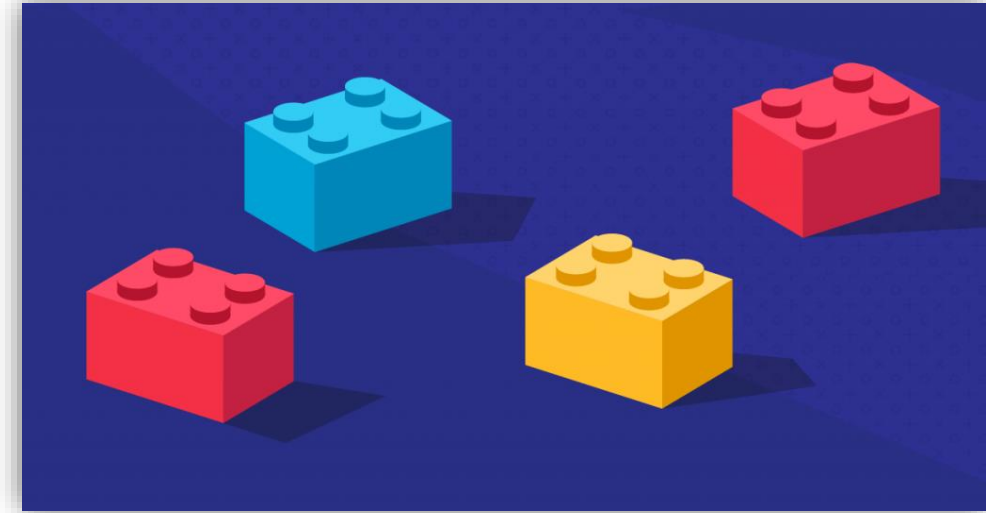
```
<h1 class="maClasse">salut l'équipe !!8</h1>
```

JSX

- ▷ Finalement le JSX est juste une manière plus simple d'écrire du HTML purement en JavaScript en évitant le fameux innerHTML
- ▷ De plus, vous pouvez être sûr que vos balises seront valides car le parseur JSX est stricte.
- ▷ Il évite aussi les failles XSS en échappant tous les caractères spéciaux

Les propriétés des composants

- ▷ Un composant peut être configuré à l'aide de propriétés
- ▷ Celles-ci peuvent être apparentées à des paramètres de fonction



```
<QuiEstLeMeilleur color='red' />
```

React c'est le plus mieux

```
<QuiEstLeMeilleur color='blue' />
```

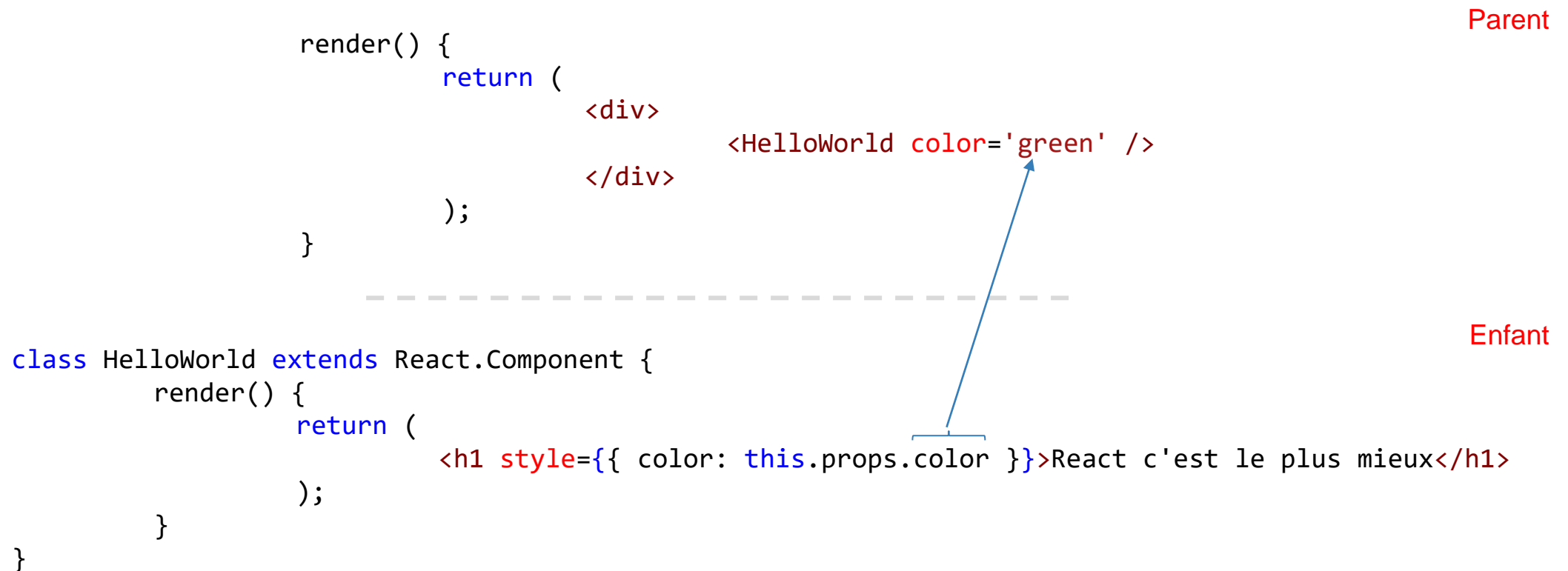
React c'est le plus mieux

```
<QuiEstLeMeilleur color='green' />
```

React c'est le plus mieux

Passage de propriétés

- Les propriétés d'un composant enfant sont accessibles au travers un objet « **props** »



Passage de propriétés - fonctions

▷ Rien ne change côté parent !

Parent

```
return (  
  <div>  
    <HelloWorld color='green' />  
  </div>  
);
```

Enfant

```
function HelloWorld (props) {  
  return (  
    <h1 style={{ color: props.color }}>React, c'est le plus mieux</h1>  
  );  
}
```



Props

- ▷ Seulement trois props sont réservées par React :
- › key: différencie les composants dans une liste à l'aide d'un identifiant unique ([en savoir plus](#))
 - › ref: permet de manipuler directement l'élément dans le DOM
 - › children: liste des composants enfants,

```
const numbers = [1, 2, 3, 4, 5];  
const listItems = numbers.map(number => (  
  <li key={number.toString()}>{number}</li>  
));
```

← Bonne pratique, chaque clé doit être unique



Etat d'un composant


```

✓ JSONSchema
1 {
2   "title": "A registration form",
3   "description": "A simple form example.",
4   "type": "object",
5   "required": [
6     "firstName",
7     "lastName"
8   ],
9   "properties": {
10    "firstName": {
11      "type": "string",
12      "title": "First name"
13    },
14    "lastName": {
15      "type": "string",

```

```

✓ UISchema
1 {
2   "firstName": {
3     "ui:autofocus": true
4   },
5   "age": {
6     "ui:widget": "updown"
7   },
8   "bio": {
9     "ui:widget": "textarea"
10  },
11  "password": {
12    "ui:widget": "password",
13    "ui:help": "Hint: Make it
14    strong!"

```

```

✓ formData
1 {
2   "firstName": "Chuck",
3   "lastName": "Norris",
4   "age": 75,
5   "bio": "Roundhouse kicking
6   asses since 1940",
7   "password": "noneed"

```

A registration form

A simple form example.

First name*

Last name*

Age

Bio

Password

Hint: Make it strong!

State

- ▷ Chaque composant peut avoir un état (ou **state**) qui lui est propre.
 - › Ce state n'est pas visible par les autres composants !
- ▷ Permet d'avoir de l'interactivité dans notre composant
- ▷ **Déclenche le réaffichage de son composant quand il est modifié**
 - › Doit être déclaré dans le constructeur du composant
- ▷ Equivalent à la propriété data dans VueJS ou \$scope dans AngularJS.

State

► Pour commencer, on doit appeler le constructeur parent avec les mêmes paramètres donnés à notre constructeur.

```
class HelloWorld extends React.Component {  
  constructor(props) {  
    super(props); // Obligatoire si on souhaite accéder à this.props dans le constructeur  
    this.state = { counter: 0 };  
  }  
  
  render() {  
    return (  
      <h1>Hello World: {this.state.counter} times</h1>  
    );  
  }  
}
```

State

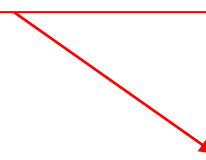
- ▷ Le state est mis à jour de manière asynchrone pour des raisons de performance. Il est modifié seulement avec la méthode `setState` d'un composant ;
- ▷ Il existe deux manières d'appeler `setState` :
 - › Statiquement: simple à écrire mais ne permet pas modifier correctement en mode batch
 - › Dynamiquement: plus verbeuse à écrire mais gère les cas plus complexes

[Plus d'infos ici](#)

State

▷ *Note: Le binding dans le constructeur est nécessaire pour notre méthode increment car elle sera appelée dans une callback où `this` fait référence à `window`*

```
class HelloWorld extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      value: 0  
    };  
  
    this.increment = this.increment.bind(this);  
  }  
  
  increment() {  
    this.setState({  
      value: this.state.value + 1  
    });  
  }  
  
  ...  
}
```




Mauvaise pratique

State

- ▷ `setState` est exécutée de manière asynchrone ;
- ▷ Pour éviter les erreurs, on préférera cette pratique :

```
class HelloWorld extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      value: 0  
    };  
  
    this.increment = this.increment.bind(this);  
  }  
  
  increment() {  
    // Modification dynamique  
    this.setState(oldState => ({  
      value: oldState.value + 1  
    }));  
  }  
  
  ...  
}
```



Callback attendu !

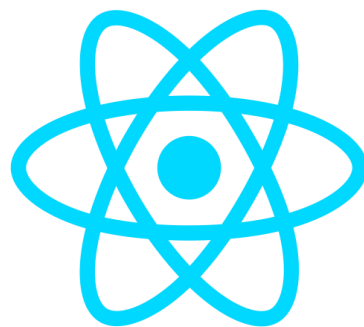
State - Affichage

```
render() {  
  // La méthode onClick réagit au click de l'utilisateur sur l'élément  
  // Attend une référence de fonction !  
  return (  
    <div>  
      <h1>Valeur actuelle du compteur : {this.state.value}</h1>  
      <p onClick={this.increment}>Cliquez ici pour incrémenter la valeur</p>  
    </div>  
  );  
}
```

Permet d'accéder aux variable de l'état actuel 😊

Peut être appliqué sur n'importe quel élément

Un composant enfant n'hérite pas du state père !



create-react-app

Créer un projet React

- ▷ Pour simplifier le développement en ReactJS, nous allons utiliser « **create-react-app** »
- ▷ Aide à la création d'un projet React
- ▷ Ce scripts va installer ces outils :
 - › Webpack, outil de build pour le Front end
 - › ESLint, linter pour JavaScript
 - › Jest, solution de testing en JavaScript préconfigurée pour React
 - › Babel, préconfiguré pour transpiler du code ES6, JSX vers du ES5

Create-react-app

- ▷ Génère une appli et un automatisateur de tâches
 - › Géré par Webpack comme vu précédemment
- ▷ Permet l'automatisation des tâches suivantes :
 - › Transpilation ES6 et JSX ;
 - › Serveur de développement avec rechargement de module à chaud ;
 - › Linting code ;
 - › Préfixe CSS ;
 - › Créer un script avec JS, CSS et regroupement d'images, et des sourcemaps ;
 - › Cadre de test Jest.

Installation

- ▷ Tout d'abord, installez **create-react-app** globalement avec le gestionnaire de packages (npm) ;
 - › NodeJS est nécessaire, téléchargez-le [ici](#)
- ▷ Ouvrez votre terminal, naviguez (*cd*) vers le répertoire où vous souhaitez installer et tapez :

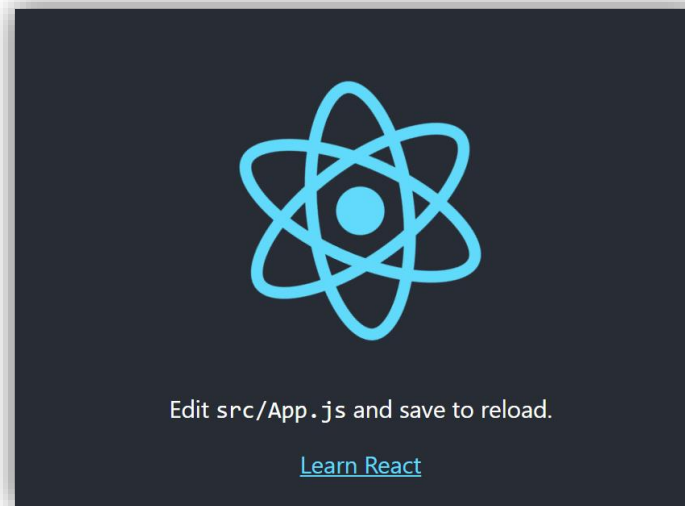
```
> npx create-react-app nom-de-mon-projet
```

Lancez votre projet

- ▷ Attendez l'installation, entrez dans le dossier et lancez :

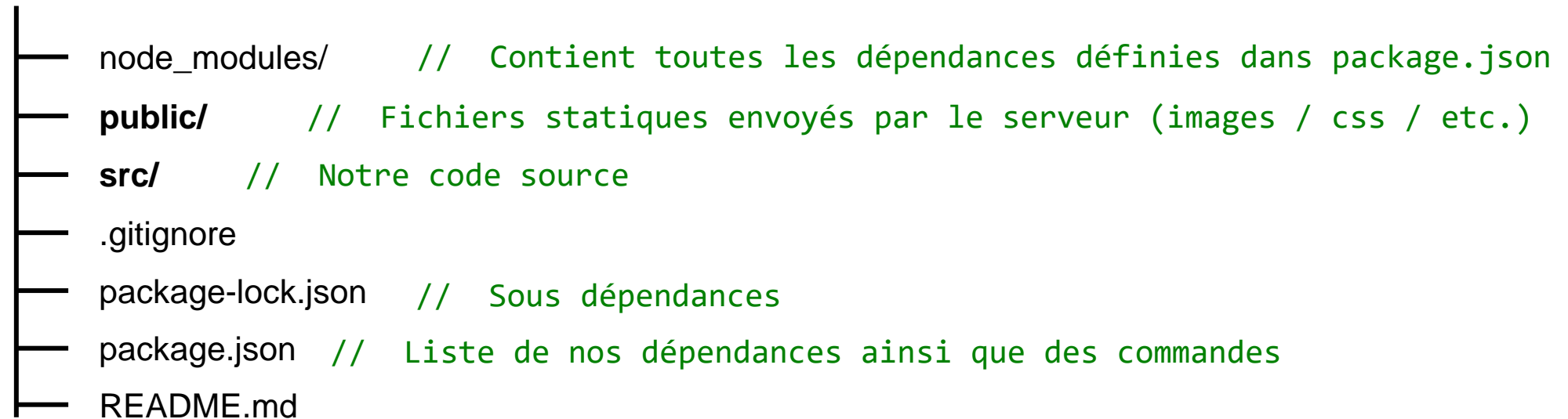
```
> npm start
```

- ▷ Et sous nos yeux ébahis



Fichiers générés

reddit-like/



Configuration

- ▷ Par défaut, create-react-app est volontairement opaque ET non configurable ;
- ▷ Parfois, on souhaite appliquer d'autres configurations
 - › utiliser un langage CSS compilé comme Sass par exemple
- ▷ Pour y arriver, on lance la commande d'éjection :

```
> npm run eject
```

- ▷ Permet ainsi de modifier les fichiers de configuration
 - › **C'est irréversible !**

Question ?