



Formation ReactJS, programmation avancée

Romain Bohdanowicz

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

<http://formation.tech/>



- Romain Bohdanowicz
Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle
- Expérience
Formateur/Développeur Freelance depuis 2006
Plus de 10 000 heures de formation animées
- Langages
Expert : HTML / CSS / JavaScript / PHP / Java
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- Certifications
PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- Divers
Premier site web à 12 ans (HTML/JS/PHP), Loisirs : Triathlon
- Et vous ?
Langages ? Expérience ? Utilité de cette formation ?



ECMAScript 6

ECMAScript 6 - Introduction



- ECMAScript 6, aussi connu sous le nom ECMAScript 2015 ou ES6 est la plus grosse évolution du langage depuis sa création (juin 2015)
<http://www.ecma-international.org/ecma-262/6.0/>
- Le langage est enfin adapté à des application JS complexes (modules, promesses, portées de blocks...)
- Pour découvrir les nouveautés d'ECMAScript 2015 / ES6
<http://es6-features.org/>

ECMAScript 6 - Compatibilité



- Compatibilité (novembre 2016) :
 - Dernière version de Chrome/Opera, Edge, Firefox, Safari : ~ 90%
 - Node.js 6 et 7 : ~ 90% d'ES6
 - Internet Explorer 11 : ~ 10% d'ES6
- Pour connaître la compatibilité des moteurs JS :
<http://kangax.github.io/compat-table/>
- Pour développer dès aujourd'hui en ES6 et exécuter le code sur des moteurs plus anciens on peut utiliser des :
 - Compilateurs ou transpileurs : Babel, Traceur, TypeScript... Transforment la syntaxe ES6 en ES5
 - Bibliothèques de polyfills : core-js, es6-shim, es7-shim... Recréent les méthodes manquante en JS

ECMAScript 6 - Portées de bloc



▸ let

- On peut remplacer le mot-clé var, par let et obtenir ainsi une portée de bloc
- La portée de bloc ainsi créée peut devenir une closure

```
for (var globalI=0; globalI<3; globalI++) {}  
console.log(typeof globalI); // number
```

```
for (let i=0; i<3; i++) {}  
console.log(typeof i); // undefined
```

```
// In 1s : 0 1 2  
for (let i=0; i<3; i++) {  
  setTimeout(() => {  
    console.log(i);  
  }, 1000);  
}
```

ECMAScript 6 - Constantes



▸ Constantes

- Il est désormais possible de créer des constantes
- Comme pour let, les variables déclarées via const ont une portée de bloc
- Bonne pratique, utiliser const ou bien let lorsque ce n'est pas possible (plus jamais var)

```
if (true) {  
  const PI = 3.14;  
}  
  
console.log(typeof PI); // undefined  
  
const hello = function() {};  
// SyntaxError: Identifier 'hello' has already been declared  
const hello = function() {};
```

ECMAScript 6 - Template literal



▸ Template literal / Template string

- Permet de créer une chaîne de caractères à partir de variables ou d'expressions
- Permet de créer des chaînes de caractères multi-lignes
- Déclarée avec un backquote ` (rarement utilisé dans une chaîne)

```
const prenom = 'Romain';
console.log(`Bonjour ${prenom} !`);

// ES5
// console.log('Bonjour ' + prenom + ' !');

const html = `
<table class="table">
  <tr><td>${prenom.toUpperCase()}</td></tr>
</table>
`;
```


ECMAScript 6 - Fonctions fléchées



▸ Arrow Functions

- Plus courtes à écrire : (params) => retour.
- Si un seul paramètre, les parenthèses des paramètres sont optionnelles.
- Si le retour est un objet, les parenthèses du retour sont obligatoires.

```
const sum = (a, b) => a + b;  
const hello = name => `Hello ${name}`;  
const getCoords = (x, y) => ({x: x, y: y});
```

```
// ES5  
// var sum = function (a, b) {  
//   return a + b;  
// };  
// var hello = function (name) {  
//   return 'Hello ' + name;  
// };  
// var getCoords = function (x, y) {  
//   return {  
//     x: x,  
//     y: y,  
//   };  
// };
```



- Avec bloc d'instructions
 - Si les fonctions nécessitent plusieurs lignes, on peut utiliser un bloc { }
 - Le mot clé return devient alors obligatoire

```
const isWon = (nbGiven, nbToGuess) => {  
  if (nbGiven < nbToGuess) {  
    return 'Too low';  
  }  
  
  if (nbGiven > nbToGuess) {  
    return 'Too high';  
  }  
  
  return 'Won !';  
};
```

ECMAScript 6 - Fonctions fléchées



▸ Bonnes pratiques

- Attention à ne pas utiliser les fonctions fléchées pour déclarer des méthodes !
- Utiliser les fonctions fléchées pour les callback ou les fonctions hors objets
- Utiliser les method properties pour les méthodes
- Utiliser class pour les fonctions constructeurs

```
const globalThis = this;

const contact = {
  firstName: 'Romain',
  method1: () => { // Mauvaise pratique
    console.log(this === globalThis); // true
  },
  method2() { // Bonne pratique
    console.log(this === contact); // true
  }
};

contact.method1();
contact.method2();
```

ECMAScript 6 - Default Params



▸ Paramètres par défaut

- Les paramètres d'entrées peuvent maintenant recevoir une valeur par défaut

```
const sum = function(a, b, c = 0) {  
  return a + b + c;  
};
```

```
console.log(sum(1, 2, 3)); // 6  
console.log(sum(1, 2)); // 3
```

```
// ES5  
// var sum = function(a, b, c) {  
//   if (c === undefined) {  
//     c = 0;  
//   }  
//   return a + b + c;  
// };
```

ECMAScript 6 - Rest Parameters



▸ Paramètres restants

- Pour récupérer les valeurs non déclarées d'une fonction on peut utiliser le REST Params
- Remplace la variable arguments (qui n'existe pas dans une fonction fléchée)
- La variable créée est un tableau (contrairement à arguments)
- Bonne pratique : ne plus utiliser arguments

```
const sum = (a, b, ...others) => {  
  let result = a + b;  
  
  others.forEach(nb => result += nb);  
  
  return result;  
};  
console.log(sum(1, 2, 3, 4)); // 10  
  
const sumShort = (...n) => n.reduce((a, b) => a + b);  
console.log(sumShort(1, 2, 3, 4)); // 10
```

ECMAScript 6 - Spread Operator



▸ Spread Operator

- Le Spread Operator permet de transformer un tableau en une liste de valeurs.

```
const sum = (a, b, c, d) => a + b + c + d;

const nbs = [2, 3, 4, 5];
console.log(sum(...nbs)); // 14
// ES5 :
// console.log(sum(nbs[0], nbs[1], nbs[2], nbs[3]));

const otherNbs = [1, ...nbs, 6];
console.log(otherNbs.join(', ')); // 1, 2, 3, 4, 5, 6
// ES5 :
// const otherNbs = [1, nbs[0], nbs[1], nbs[2], nbs[3], 6];

// Clone an array
const cloned = [...nbs];
```

ECMAScript 6 - Shorthand property



- Shorthand property

- Lorsque l'on affecte une variable à une propriété (maVar: maVar), il suffit de déclarer la propriété

```
const x = 10;  
const y = 20;  
  
const coords = {  
  x,  
  y,  
};  
  
// ES5  
// const coords = {  
//   x: x,  
//   y: y,  
// };
```

ECMAScript 6 - Method properties



- Method properties
 - Syntaxe simplifiée pour déclarer des méthodes

```
const maths = {  
  sum(a, b) {  
    return a + b;  
  }  
};  
  
console.log(maths.sum(1, 2)); // 3  
  
// ES5  
// const maths = {  
//   sum: function(a, b) {  
//     return a + b;  
//   }  
// };
```


ECMAScript 6 - Computed Property Names



▸ Computed Property Names

Permet d'utiliser une expression en nom de propriété

```
let i = 0;

const users = {
  [`user${++i}`]: { firstName: 'Romain' },
  [`user${++i}`]: { firstName: 'Steven' },
};

console.log(users.user1); // { firstName: 'Romain' }
```



```
/* ES5
var i = 0;
var users = {};
users['user ' + (++i)] = { firstName: 'Romain' };
users['user ' + (++i)] = { firstName: 'Steven' };

console.log(users.user1); // { firstName: 'Romain' }
*/
```

ECMAScript 6 - Array Destructuring



▸ Déstructurer un tableau

- Permet de déclarer des variables recevant directement une valeur d'un tableau

```
//      [1  , 2  , 3  ];  
const [one, two, three] = [1, 2, 3];  
console.log(one); // 1  
console.log(two); // 2  
console.log(three); // 3  
  
// ES5  
// var tmp = [1, 2, 3];  
// var one = tmp[0];  
// var two = tmp[1];  
// var three = tmp[2];
```

ECMAScript 6 - Array Destructuring



- Déstructurer un tableau
 - Il est possible de ne pas déclarer un variable pour chaque valeur
 - Il est possible d'utiliser une valeur par défaut
 - Il est possible d'utiliser le REST Params

```
const [one, , three = 3] = [1, 2];  
console.log(one); // 1  
console.log(three); // 3  
  
const [romain, ...others] = ['Romain', 'Jean', 'Eric'];  
console.log(romain); // Romain  
console.log(others.join(', ')); // Jean, Eric
```

ECMAScript 6 - Object Destructuring



- Déstructurer un objet

- Comme pour les tableaux il est possible de déclarer une variable recevant directement une propriété

```
//      {x: 10  , y: 20  }  
const {x: varX, y: varY} = {x: 10, y: 20};  
console.log(varX); // 10  
console.log(varY); // 20
```

ECMAScript 6 - Object Destructuring



- Déstructurer un objet
 - Il est possible de nommer sa variable comme la propriété et d'utiliser shorthand property
 - Il est possible d'utiliser une valeur par défaut

```
const {x: x , y , z = 30} = {x: 10, y: 20};  
console.log(x); // 10  
console.log(y); // 20  
console.log(z); // 30
```

ECMAScript 6 - Mot clé class



- Simplifie la déclaration de fonction constructeur
- Les classes n'existent pas pour autant en JavaScript, ce n'est qu'une syntaxe simplifiée (sucre syntaxique)
- Le contenu d'une classe est en mode strict

```
class Person {  
  constructor(firstName) {  
    this.firstName = firstName;  
  }  
  hello() {  
    return `Hello my name is ${this.firstName}`;  
  }  
}  
  
const instructor = new Person('Romain');  
console.log(instructor.hello()); // Hello my name is Romain  
  
// ES5  
// var Person = function(firstName) {  
//   this.firstName = firstName;  
// };  
// Person.prototype.hello = function() {  
//   return 'Hello my name is ' + this.firstName;  
// };
```

ECMAScript 6 - Mot clé class



- Héritage avec le mot clé class
 - Utilisation du mot clé `extends` pour l'héritage
 - Utilisation de `super` pour appeler la fonction constructeur parent et les accès aux méthodes parents si redéclarée dans la classe

```
class Instructor extends Person {  
  constructor(firstName, speciality) {  
    super(firstName);  
    this.speciality = speciality;  
  }  
  hello() {  
    return `${super.hello()}, my speciality is ${this.speciality}`;  
  }  
}  
  
const romain = new Instructor('Romain', 'JavaScript');  
console.log(romain.hello()); // Hello my name is Romain, my speciality is  
JavaScript
```



Modules ECMAScript

Modules ECMAScript - Introduction



- JavaScript à sa conception
 - Objectif : créer des interactions côté client, après chargement de la page
 - Exemples de l'époque :
 - Menu en rollover (image ou couleur de fond qui change au survol)
 - Validation de formulaire
- JavaScript aujourd'hui
 - Applications front-end, back-end, en ligne de commande, de bureau, mobiles...
 - Applications pouvant contenir plusieurs centaines de milliers de lignes de codes (Front-end de Facebook > 1 000 000 LOC)
 - Il faut faciliter le travail collaboratif, en plusieurs fichiers et en limitant les risques de conflit

Modules ECMAScript - Introduction

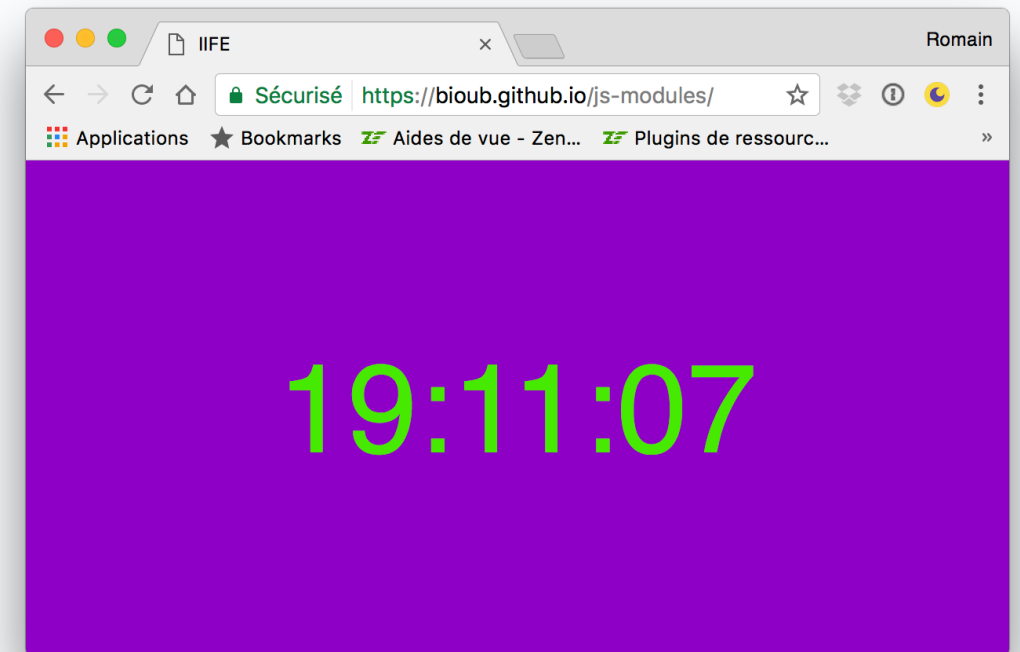
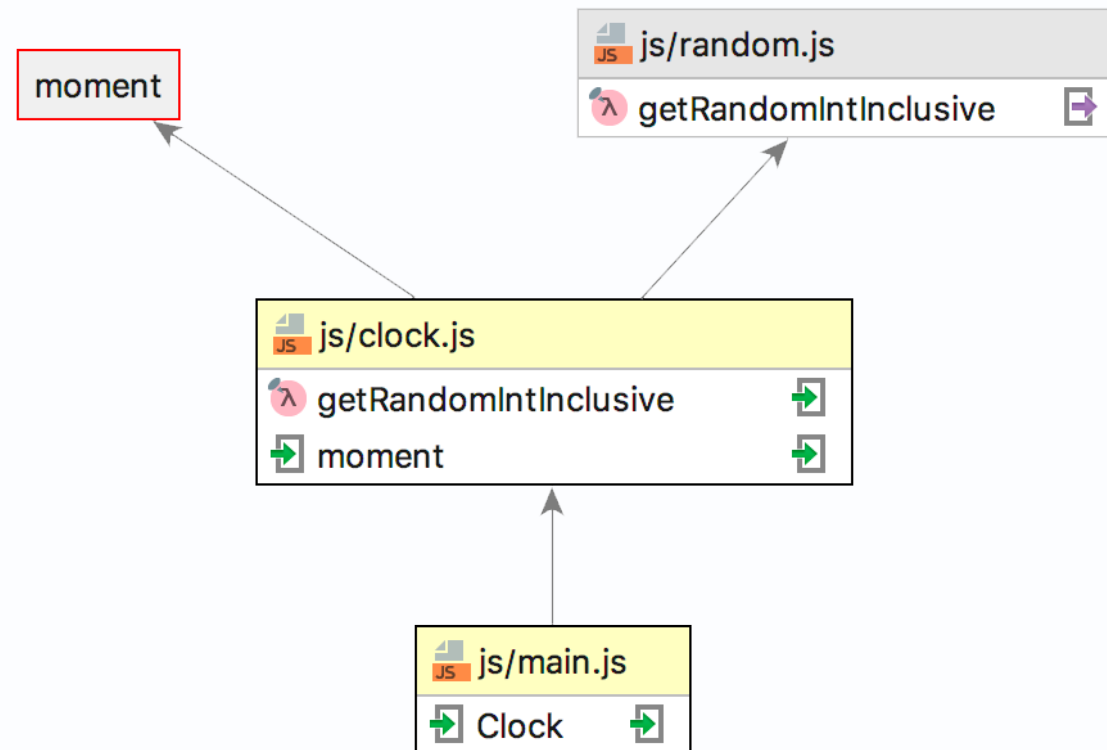


- Objectifs d'un module JavaScript
 - Créer une portée au niveau du fichier
 - Permettre l'export et l'import d'identifiants (variables, fonctions...) entre ces fichiers qui auront désormais leur propre portée
- Principaux systèmes existants
 - IIFE / Function Wrapper
 - CommonJS
 - AMD
 - UMD
 - SystemJS
 - ES6 (statiques mots clés import / export)
 - ESNext : import() (fonction asynchrone)

Modules ECMAScript - Introduction



- Exemple utilisé pour la suite



- Le point d'entrée de l'application est le fichier main.js, qui dépend de Clock défini dans le fichiers clock.js, qui dépend lui même de getRandomIntInclusive du fichier random.js et moment définit dans le projet Open Source Moment.js
- Exemples : <https://github.com/bioub/js-modules/>
- Démo : <https://bioub.github.io/js-modules/>



- Portée de modules
 - Sans module la portée d'une fonction ou d'une variable déclarée dans un fichier serait globale.
 - Avec les modules une fonction sera locale au fichier.
- Import / Export
 - Une fonction ou un objet pouvant servir dans un autre fichier il faudra l'exporter.
 - Cela va créer l'API public du fichier (accessible de l'extérieur).
 - Un autre fichier devra importer les fonctions utilisées
- Mode Strict
 - Les modules ECMAScript sont par défaut en mode strict, il n'est donc pas nécessaire d'écrire *'use strict'*; en début de fichier.

Module ECMAScript - Export



- Pour exporter une variable ou une fonction on utilise le mot clé export

```
export const getRandom = function() {  
  return Math.random();  
};  
  
export const getRandomArbitrary = function(min, max) {  
  return Math.random() * (max - min) + min;  
};  
  
export const getRandomInt = function(min, max) {  
  min = Math.ceil(min);  
  max = Math.floor(max);  
  return Math.floor(Math.random() * (max - min)) + min;  
};  
  
export const getRandomIntInclusive = function(min, max) {  
  min = Math.ceil(min);  
  max = Math.floor(max);  
  return Math.floor(Math.random() * (max - min + 1)) + min;  
};
```



- Il est également possible d'exporter en une seule fois en fin de fichier

```
const getRandom = function() {  
    return Math.random();  
};  
  
const getRandomArbitrary = function(min, max) {  
    return Math.random() * (max - min) + min;  
};  
  
const getRandomInt = function(min, max) {  
    min = Math.ceil(min);  
    max = Math.floor(max);  
    return Math.floor(Math.random() * (max - min)) + min;  
};  
  
const getRandomIntInclusive = function(min, max) {  
    min = Math.ceil(min);  
    max = Math.floor(max);  
    return Math.floor(Math.random() * (max - min + 1)) + min;  
};  
  
export { getRandom, getRandomArbitrary, getRandomInt, getRandomIntInclusive };
```

Module ECMAScript - Import



- Pour importer on utilise le mot clé `import`, associé à des accolades et le nom du fichier (l'extension est optionnelle)
- Lorsque que le fichier fait partie du projet, il est obligatoire de préfixer le fichier par `./` ou `../`
- Les modules ECMAScript ne peuvent être importée que statiquement en début de fichier. Pour des imports dynamiques il faut utiliser les modules CommonJS ou Dynamic Import (ESNext)

```
import { getRandomIntInclusive } from './random';

class Clock {
  // ...

  update() {
    let r = getRandomIntInclusive(0, 255);
    let g = getRandomIntInclusive(0, 255);
    let b = getRandomIntInclusive(0, 255);
    // ...
  }

  // ...
}
```

Module ECMAScript - Tree Shaking



- Les imports étant statiques, des bundlers (bibliothèques de build) comme webpack ou Rollup peuvent analyser le code et éliminer du build les exports non importés
- Le build final ressemblera ainsi à :

```
const getRandomIntInclusive = function(min, max) {  
  min = Math.ceil(min);  
  max = Math.floor(max);  
  return Math.floor(Math.random() * (max - min + 1)) + min;  
};  
  
class Clock {  
  // ...  
  
  update() {  
    let r = getRandomIntInclusive(0, 255);  
    let g = getRandomIntInclusive(0, 255);  
    let b = getRandomIntInclusive(0, 255);  
    // ...  
  }  
  
  // ...  
}
```


Module ECMAScript - Export/Import par défaut



- Il est possible de définir un export par défaut lorsqu'on a qu'une seule valeur à importer ou une valeur principale à importer
- Pour exporter on ajoute le mot clé *default*

```
export default class Clock {  
    // ...  
}
```

- Pour importer il faudra ne pas utiliser d'accolades

```
import Clock from './clock';  
  
let clockElt = document.querySelector('.clock');  
let clock = new Clock(clockElt);  
clock.start();
```

- Certains développeurs conseillent d'éviter les exports par défaut :
<https://basarat.gitbooks.io/typescript/docs/tips/defaultIsBad.html>

Module ECMAScript - Imports avancés



- On peut renommer un import, par exemple dans le cas où 2 identifiants auraient le même nom :

```
import { render as renderDom } from 'react-dom';  
import { App } from './App';  
  
renderDom(<App />, document.getElementById('root'));
```

- On peut également importer tous les exports dans un objet :

```
// serviceWorker.js  
export function register(config) {  
  // ...  
}  
  
export function unregister() {  
  // ...  
}
```

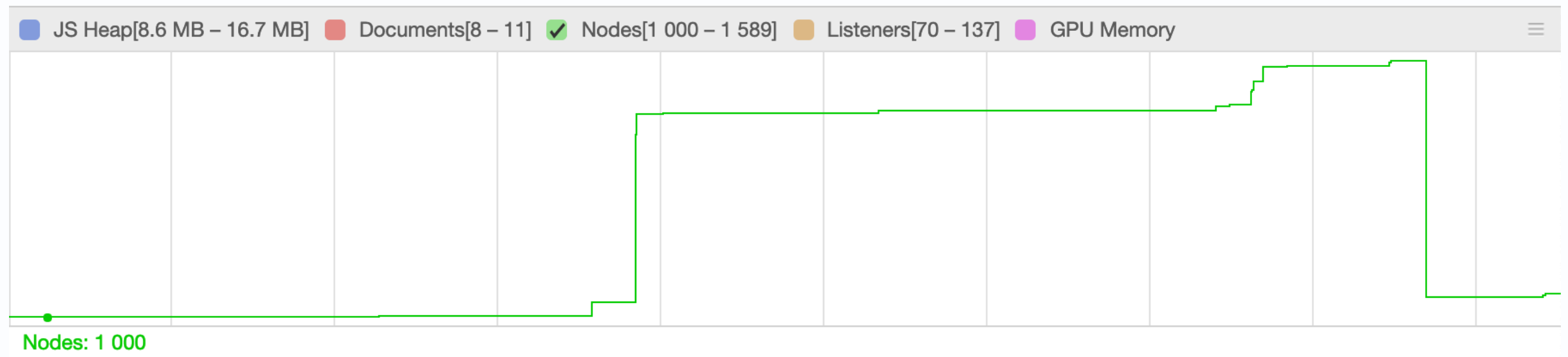
```
import * as serviceWorker from './serviceWorker';  
  
serviceWorker.unregister();
```



React



- Le DOM ou Document Objet Model est l'API du navigateur créé par Netscape en 1995 qui permet de manipuler le contenu de la page web
- Cet API est ancien même s'il reçoit des évolutions régulièrement
- Il est également très lourd, par exemple la page d'accueil de formation.tech va créer jusqu'à 1589 objet associés au DOM en mémoire



- Lorsqu'un composant React doit se rafraîchir (en appelant sa méthode render), il serait très coûteux de recréer tout les éléments du DOM qu'il contient. Pour éviter cela React met en place un "Virtual DOM"



- Voici un exemple de mini-framework sans Virtual DOM

```
class Component {
  _refresh() {
    this.host.innerHTML = '';
    this.render().forEach(elt => this.host.appendChild(elt));
  }
  setState(newState) {
    Object.assign(this.state, newState);
    this._refresh();
  }
}

function domRender(component, host) {
  component.host = host;
  component._refresh();
}
```

- Comme dans React, appeler la méthode `setState` ou `domRender` provoquera le rafraîchissement du composant en appelant sa méthode `render`.



- Comme React, le composant possède une méthode render qui construit le DOM

```
class ButtonCount extends Component {
  state = { count: 0 };
  increment = () => {
    this.setState({count: this.state.count + 1});
  };
  render() {
    const p = document.createElement('p');
    p.innerText = 'Démo : ';

    const button = document.createElement('button');
    button.innerText = this.state.count;
    button.onclick = this.increment;

    return [p, button];
  }
}

domRender(new ButtonCount(), document.querySelector('hello-component'));
```

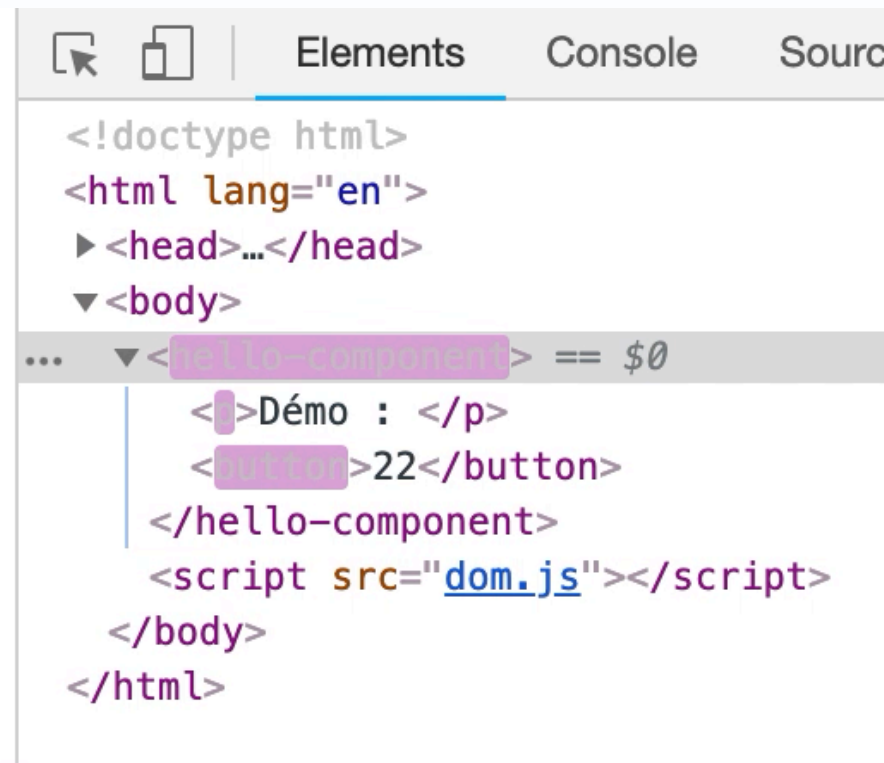
- On remarque que l'API DOM est lourd, si on pouvait chainer comme jQuery il n'y aurait que 2 lignes dans render
- Puis on peut demander le rendu dans une balise existante ici hello-component



- Lorsqu'on observe le résultat avec les DevTools de Chrome, on voit que l'ensemble du DOM associé au composant est rafraîchi

Démo :

22



```
<!doctype html>
<html lang="en">
  <head>...</head>
  <body>
    ... <hello-component> == $0
      <p>Démo : </p>
      <button>22</button>
    </hello-component>
    <script src="dom.js"></script>
  </body>
</html>
```



- Avec `React.createElement` on va construire avec un API plus moderne un arbre léger en mémoire appelé Virtual DOM

```
class ButtonCount extends React.Component {
  state = { count: 0 };
  increment = () => {
    this.setState({count: this.state.count + 1});
  };
  render() {
    return [
      React.createElement('p', null, 'Démo : '),
      React.createElement('button', {onClick: this.increment}, this.state.count),
    ];
  }
}

ReactDOM.render(
  React.createElement(ButtonCount),
  document.querySelector('hello-component'),
);
```




- ▶ Avec React et son Virtual DOM on remarque que le navigateur ne rafraîchit pas plus d'élément que nécessaire :

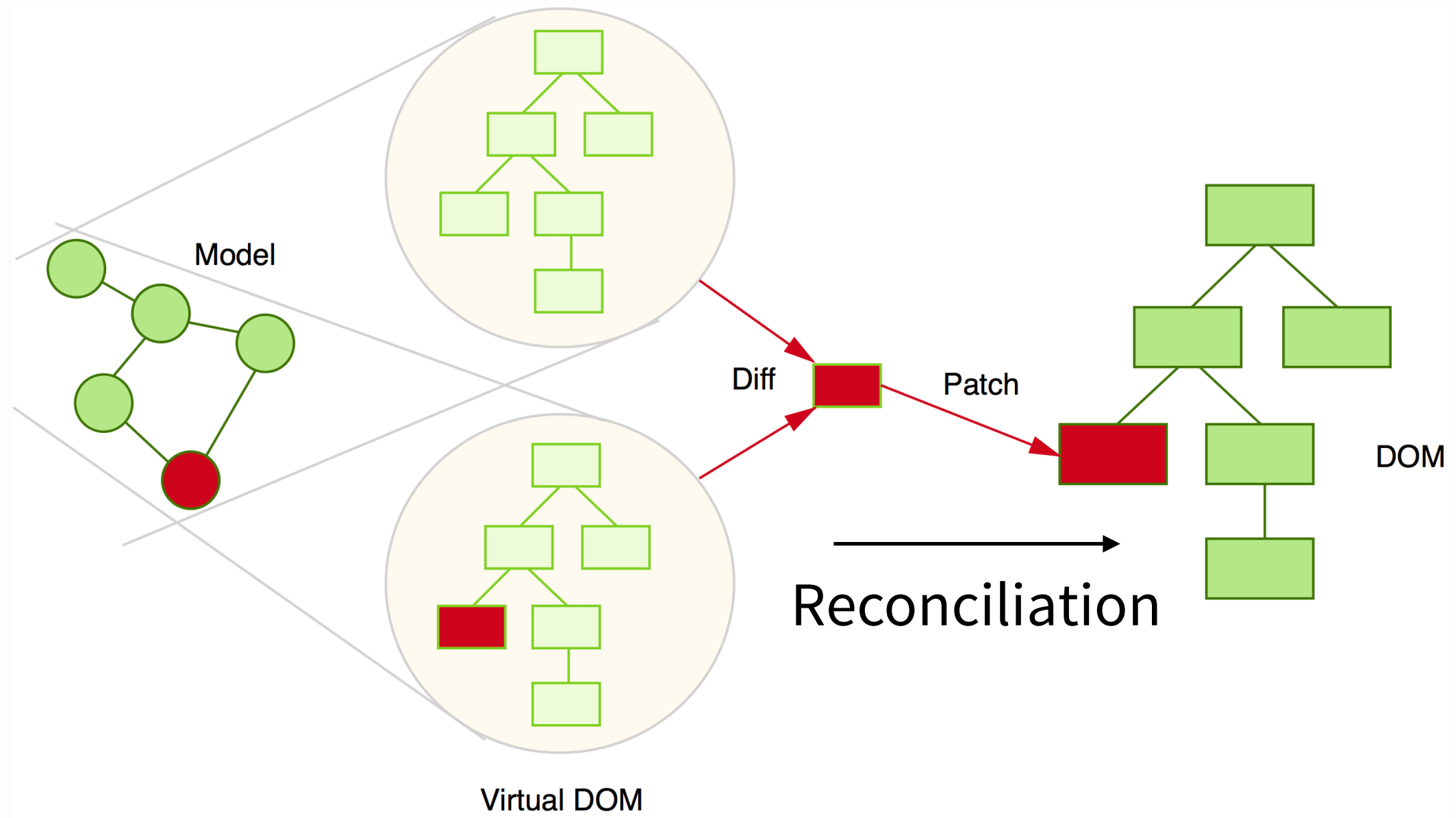
Démo :



```
Elements Console Source
<!doctype html>
<html lang="en">
  > <head>...</head>
  ▼ <body>
    ... ▼ <hello-component> == $0
      <p>Démo : </p>
      <button>22</button>
    </hello-component>
    <script src="../node_modules/react"
    <script src="../node_modules/react
    <script src="react.js"></script>
  </body>
```



▸ Réconciliation





- L'exemple précédent est encore trop verbeux. Afin de le simplifier, React a créé une syntaxe appelée JSX pour construire le Virtual DOM d'un composant
- Le navigateur ne reconnaissant pas cette syntaxe on va utiliser un compilateur (en général Babel et son plugin `@babel/plugin-transform-react-jsx`) pour transformer le JSX en `React.createElement`

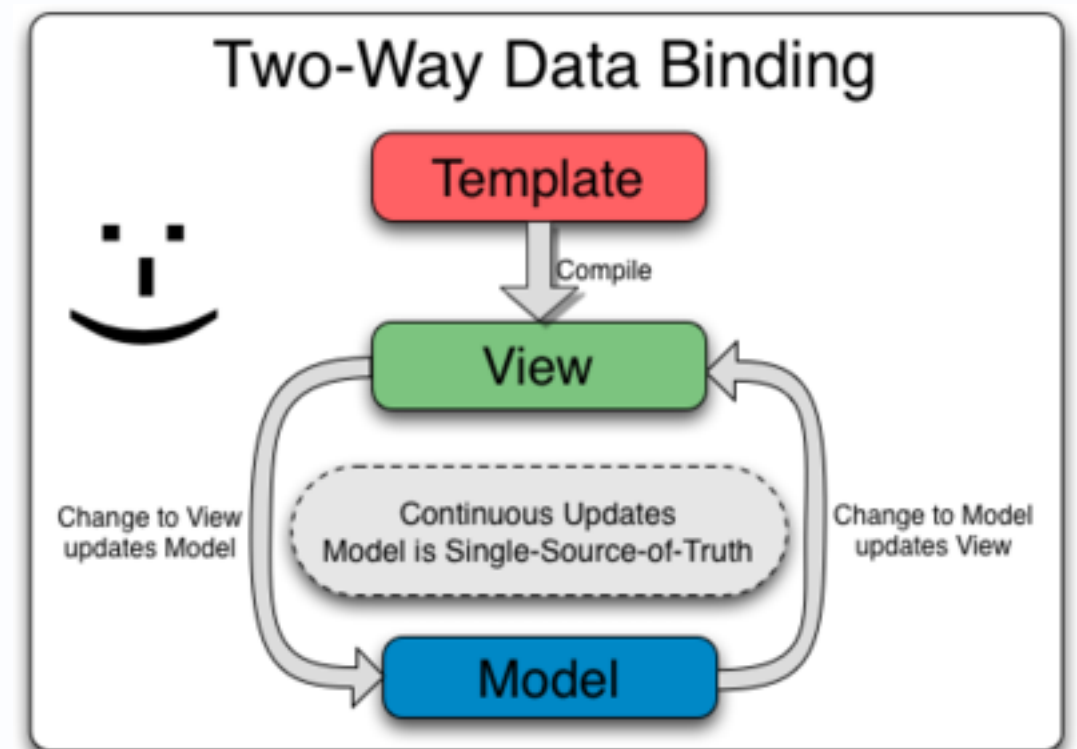
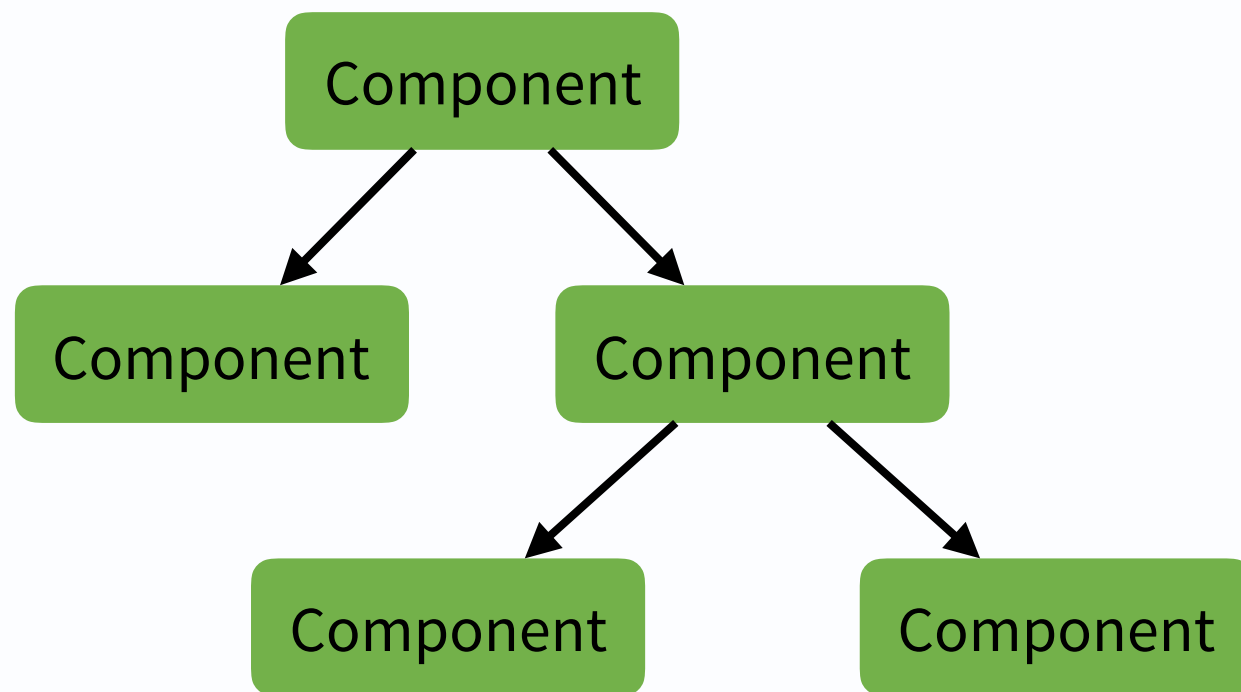
```
class ButtonCount extends React.Component {
  state = { count: 0 };
  increment = () => {
    this.setState({count: this.state.count + 1});
  };
  render() {
    return [
      <p>Démo :</p>,
      <button onClick={this.increment}>{this.state.count}</button>,
    ];
  }
}

ReactDOM.render(
  <ButtonCount />,
  document.querySelector('hello-component'),
);
```

React - One Way Data Flow



- Par opposition aux frameworks de génération précédente comme AngularJS, Knockout ou Ember.js, les données circulent toujours dans un sens dans React : d'un composant parent vers un composant enfant. On parle de One-Way Data Flow, One-Way Data Binding ou Unidirectional Data Flow





- React Developer Tools

- Extension officielle de Facebook
- Fonctionne avec Chrome et Firefox
- Permet de surveiller les objets props, state, context

- Téléchargement

- Chrome

<https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi>

- Firefox

<https://addons.mozilla.org/fr/firefox/addon/react-devtools/>

React - Outils de debug



React DevTools interface showing the React component tree and props.

React Component Tree:

```
<Provider>
  <Context.Provider>
    <HashRouter hashType="noslash">
      <Router>
        <App>
          <div className="App">
            <h3>Faire un essai gratuit</h3>
            <Route path="/" exact={true}></Route>
            <Route path="/step2"></Route>
            <Route path="/step3">
              <Step3...</Step3> == $r
            </Route>
            <Route path="/step4"></Route>
            <Route path="/step5"></Route>
            <Route path="/step6"></Route>
          </div>
        </App>
      </Router>
    </HashRouter>
  </Context.Provider>
</Provider>
```

Props for the selected component (Step3):

- history: {...}
 - action: "POP"
 - block: block()
 - createHref: createHref()
 - go: go()
 - goBack: goBack()
 - goForward: goForward()
 - length: 4
 - listen: listen()
 - location: {...}
 - push: push()
 - replace: replace()
- location: {...}
 - hash: ""
 - pathname: "/step3"
 - search: ""
- match: {...}
 - ☒ isExact: true
 - params: {...}
 - path: "/step3"
 - url: "/step3"

Breadcrumbs: Provider > Context.Provider > HashRouter > Router > App > div > Route > Step3

React - Outils de debug



```
<Route path= /step2 ></Route>
▼ <Route path="/step3">
  ▶ <Step3>...</Step3> == $r
</Route>
<Route path="/step4"></Route>
<Route path="/step5"></Route>
<Route path="/step6"></Route>
</div>
</App>
</Router>
</HashRouter>
</Context.Provider>
</Provider>
```

Provider Context.Provider HashRouter Router App div Route Step3

⋮ Console What's New

▶ 🔍 top ▼ 👁 Filter Default levels ▼

```
> $r
< ▼Route {props: {...}, context: {...}, refs: {...}, updater: {...}, state: {...}, ...} ⓘ
  ▶ context: {router: {...}}
  ▶ props: {path: "/step3", component: f}
  ▶ refs: {}
  ▶ state: {match: {...}}
```



- Redux Developer Tools
 - Fonctionne avec Chrome et Firefox
 - Permet de visualiser le store, dispatcher des actions, faire du time-travel debug
- Téléchargement
 - Chrome
<https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklieibfkpmmfibljd?hl=fr>
 - Firefox
<https://addons.mozilla.org/fr/firefox/addon/reduxdevtools/>

React - Outils de debug



The screenshot displays the Redux DevTools interface, which is used for debugging state changes in a Redux application. The interface is divided into several panels:

- Top Panel:** Shows the browser address bar with the URL `http://localhost:3000/#extension-demo`.
- Left Panel:** Contains the Redux state tree. It shows the initial state `state: {}` and subsequent updates where the `counter` key is added and incremented. The state tree is expanded to show the `counter` value.
- Center Panel:** Displays the Redux state changes as a sequence of actions. The actions are: `@@@INIT`, `INCREMENT_COUNTER`, `INCREMENT_COUNTER`, `DECREMENT_COUNTER`, and `INCREMENT_COUNTER`. The state changes are visualized as a sequence of values: `counter (init): 1` and `counter (init): 2`.
- Right Panel:** Shows the Redux state tree after the actions. It displays the state after each action, showing the `counter` value being updated. The state tree is expanded to show the `counter` value.
- Bottom Panel:** Contains the Redux state tree after the actions. It displays the state after each action, showing the `counter` value being updated. The state tree is expanded to show the `counter` value.

The interface also includes a **Log monitor** panel on the right, which shows the sequence of actions and state changes. The **Inspector** panel at the bottom shows the Redux state tree after the actions, with the `counter` value being updated.



Typage statique React

Typage statique React - Introduction



- Typage statique vs typage dynamique
JavaScript contrairement à d'autres langages n'offre pas la possibilité de typer statiquement ses variables ou fonctions.
- Pourquoi typer statiquement ?
 - Autocomplétion dans les IDEs modernes (Visual Studio Code, Webstorm...)
 - Détection statique des erreurs dans l'IDE / à la compilation
- Pourquoi type dynamiquement ?
 - Flexibilité, une même instruction / fonction peut-être réutilisée pour plusieurs types
 - Temps de développement, pas avoir à définir statiquement le types des objets par exemple

Typage statique React - JSDoc



- Les commentaires JSDoc sont bien reconnus par les IDEs modernes (VSCode, Webstorm) voir <http://usejsdoc.org/>
- Un commentaire JSDoc commence par 2 étoiles `/**` commentaire `*/`
- Typer une fonction :

```
/**
 * Reducer of todos
 * @param {object[]} previousState
 * @param {object} action
 * @param {string} action.type
 * @param {object} action.payload
 * @param {number} action.payload.id
 * @param {string} action.payload.text
 * @param {boolean} action.payload.completed
 */
function todosReducer(previousState = [], { type, payload }) {
  switch (type) {
    case TODO_ADD:
      return [...previousState, payload];
    default:
      return {
        completed: (property) completed: boolean ⓘ
        id
        text
      }
  }
}
```

Typage statique React - JSDoc



- Pour typer les paramètres d'entrées et de retour d'une fonction

```
/**
 * Reducer of todos
 * @param {object[]} previousState
 * @param {object} action
 * @param {string} action.type
 * @param {object} action.payload
 * @param {number} action.payload.id
 * @param {string} action.payload.text
 * @param {boolean} action.payload.completed
 * @returns {object[]}
 */
function todosReducer(previousState = [], { type, payload }) {
  switch (type) {
    case TODO_ADD:
      return [...previousState, payload];
    default:
      return previousState;
  }
}
```

Typage statique React - JSDoc



- Définir des types réutilisables

```
/**
 * @typedef Todo
 * @property {number} id
 * @property {string} text
 * @property {boolean} completed
 */

/**
 * @typedef TodoAction
 * @property {string} type
 * @property {Todo} payload
 */

/**
 * Reducer of todos
 * @param {Todo[]} previousState
 * @param {TodoAction} action
 * @returns {Todo[]}
 */
function todosReducer(previousState = [], { type, payload }) {
  switch (type) {
    case TODO_ADD:
      return [...previousState, payload];
    default:
      return previousState;
  }
}
```

- Typen des variables

```
/** @type {string[]} items */  
let items = this.props.items;
```

```
/** @type {string[]} items */  
let items = this.props.items;
```

```
items = items.map(item => {
  const classes = classNames(css.item, {
    [css.itemSelected]: item. === selected
  });
});
```

Typage statique React - JSDoc



- ▶ Importer des types provenant d'autres fichiers

```
/** @type {import('webpack').Configuration} */  
const config = {};
```

- Nécessite d'écrire le code importé en TypeScript ou de créer des interfaces TypeScript supplémentaires.
- Le projet DefinitelyTyped permet de trouver des interfaces TypeScript pour la plupart des projets open-source :
<https://github.com/DefinitelyTyped/DefinitelyTyped>
(Dans le top 10 des projets en nombre de contributeurs <https://octoverse.github.com/projects#repositories>)

```
/** @type {import('webpack').Configuration} */
const config = {
  amd: (property) webpack.Configuration.a
  bail
  cache

```


Typage statique React - PropTypes



- Pour typer des composants React on peut utiliser PropTypes
- Inclus dans React jusqu'à la version 15, dans un paquet npm séparé depuis la 16
- Installation
npm install prop-types

```
import React from 'react';
import { string } from 'prop-types';

function Hello({name}) {
  return (
    <div className="Hello">
      Hello {name}
    </div>
  );
}



Hello.propTypes = {
  name: string,
};

export { Hello };
```

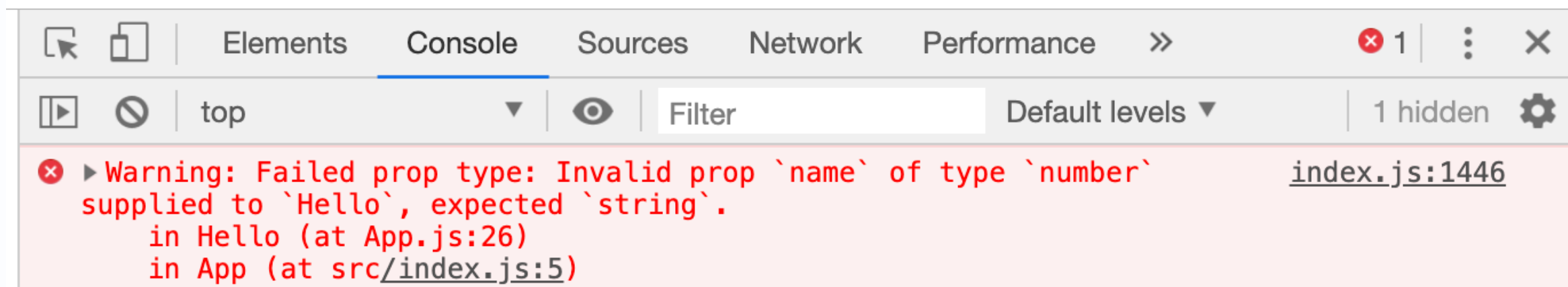
Typage statique React - PropTypes



- Complétion améliorée depuis le JSX :

```
<Hello na| />  
<Clock />  name? (JSX attribute) name?: string 
```

- Warning dans les DevTools du navigateur si on passe le mauvais type



Typage statique React - PropTypes



- Il est possible également de valider avec `isRequired` et de définir ses propres validateurs :

```
Contact.propTypes = {  
  name: PropTypes.string.isRequired,  
  age(props, propName, component) {  
    if (props[propName] && (props[propName] < 0 || props[propName] > 120)) {  
      return new Error(`${propName} should be between 0 and 120`)  
    }  
  },  
};
```

- Documentation :
<https://github.com/facebook/prop-types>
- Airbnb propose aussi ses validateurs :
<https://github.com/airbnb/prop-types>

Typage statique React - Flow



- Les commentaires JSDoc ne préviennent pas d'erreur potentielles, PropTypes affiche des warnings au moment de l'exécution
- Pour détecter statiquement des erreurs dans l'IDE ou au moment du build, Facebook propose un analyseur de type statique appelé Flow
- Flow est supporté par Create React App

Typage statique React - Flow



- Installation

```
npm i flow-bin -D
```

- Création d'un script dans le fichier package.json

```
"scripts": {  
  "start": "react-scripts start",  
  "build": "react-scripts build",  
  "flow": "flow"  
}
```

- Création du fichier de configuration

```
npm run flow init
```

- Installer l'extension VSCode Flow-Language-Support

- Désactiver la validation JavaScript de VSCode :

```
{  
  "files.autoSave": "onFocusChange",  
  "javascript.validate.enable": false  
}
```

Typage statique React - Flow



- Pour activer Flow il suffit ensuite d'utiliser le commentaire @flow

```
// @flow
function square(n: number): number {
  return n * n;
}

square("2"); // Error!
```

Typage statique React - Flow



- Typage des objets (? pour les propriétés optionnelles)

```
function Hello({name = ''}: {name: ?string}) {  
  return (  
    <div className="Hello">  
      Hello {name}  
    </div>  
  );  
}
```

- Typage avec des interfaces

```
interface HelloProps {  
  name: ?string;  
}  
  
function Hello({name = ''}: HelloProps) {  
  return (  
    <div className="Hello">  
      Hello {name}  
    </div>  
  );  
}
```

Typage statique React - TypeScript



- Create React App inclus le support de TypeScript depuis la version 2.1
- A la création du projet
`create-react-app mon-projet --typescript`
- Avantages
 - Language avec concepts supplémentaires (public/private/protected/décorateurs...)
 - Intégrations avec des bibliothèques TypeScript ou les fichiers DefinitelyTyped (imports de types...) / Intégration avec les IDEs
 - Popularité : <https://www.npmtrends.com/flow-bin-vs-typescript>
- Inconvénients
 - Flow peut s'appliquer qu'à certains fichiers
 - Flow peut s'utiliser sous forme de commentaire
 - Plus lourd à intégrer à un projet existant
- <https://github.com/niieani/typescript-vs-flowtype>

Typage statique React - Exercice



- Récupérer le projet todo-redux sur <https://gitlab.com/react-avance>
- Installer les dépendances
- Ajouter les PropTypes sur les composants TodoForm, TodoList et TodoItem
- Ajouter les Annotations Flow dans tous les fichiers du dossier src

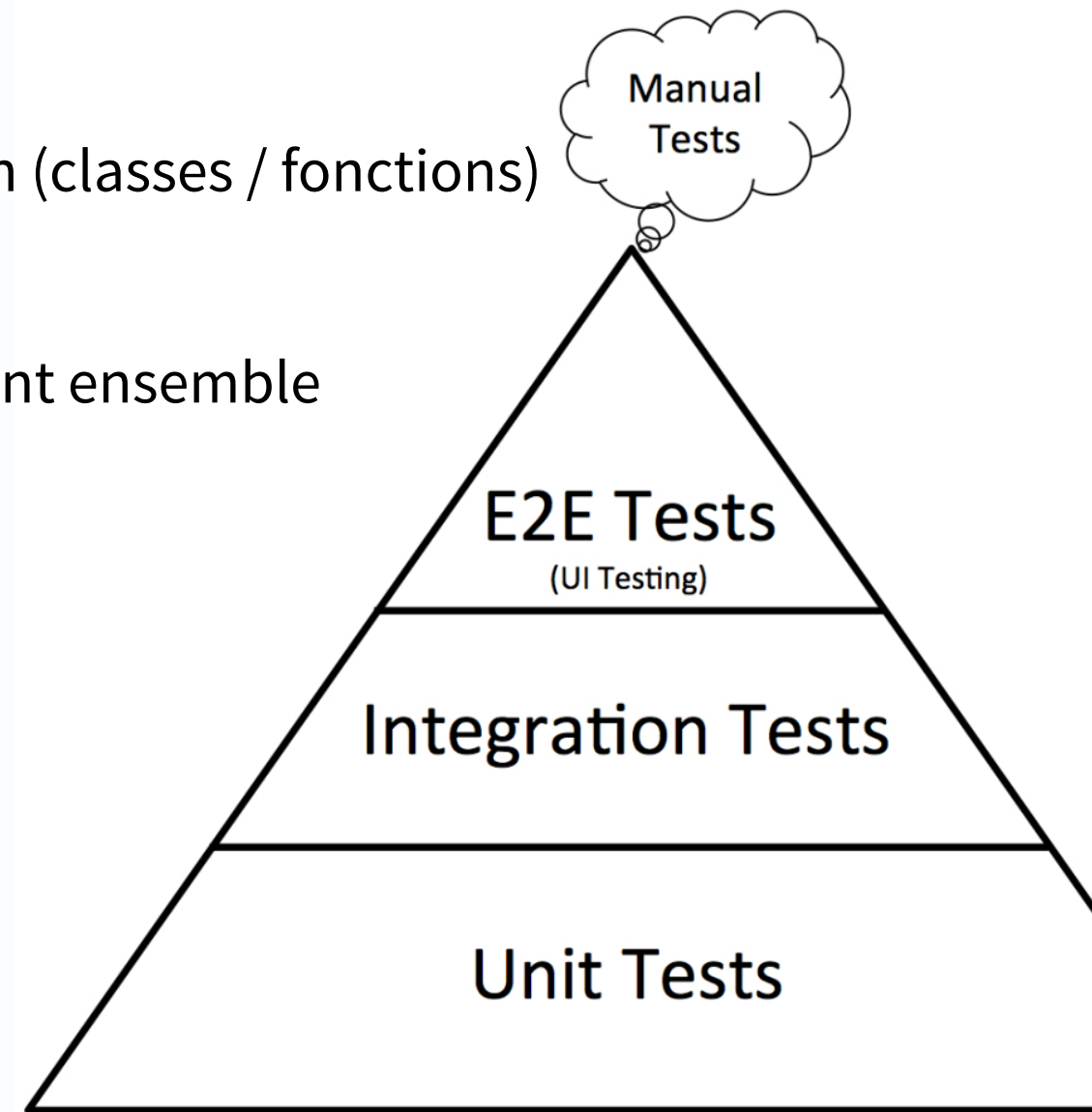


Tests avec Jest

Tests avec Jest - Introduction



- Avec les tests automatisés, les scénarios de tests sont codés et peuvent être rejoués rapidement plus régulièrement.
- 3 types de tests automatisés au niveau code côté Front :
 - Test unitaire
Permet de tester les briques d'une application (classes / fonctions)
 - Test d'intégration
Teste que les briques fonctionnent correctement ensemble
 - Test End-to-End (E2E)
Vérifie l'application dans le client



Tests avec Jest - Introduction



- Framework de test créé en 2014 par Facebook
- Sous Licence MIT depuis septembre 2017
- Permet de lancer des tests :
 - unitaires / d'intégration (dans Node.js)
 - fonctionnels / E2E (via Puppeteer)
- Peut s'utiliser avec ou sans configuration
- Les tests se lancent en parallèle dans les Workers Node.js
- Intègre par défaut :
 - Calcul de coverage (via Istanbul)
 - Mocks (natifs ou en installant Sinon.JS)
 - Snapshots

Tests avec Jest - Installation



- Installation

- `npm install --save-dev jest`

- `yarn add --dev jest`

- Déjà intégré à Create React App

Tests avec Jest - Hello, world !



- Sans configuration, les tests doivent se trouver dans un répertoire `__tests__`, ou bien se nommer `*.test.js` ou `*.spec.js`

```
// src/hello.js
const hello = (name = 'World') => `Hello ${name} !`;

module.exports = hello;
```

```
// __tests__/hello.js
const hello = require('../src/hello');

test('Hello, world !', () => {
  expect(hello()).toBe('Hello World !');
  expect(hello('Romain')).toBe('Hello Romain !');
});
```

Tests avec Jest - Lancements des tests



- Si Jest localement
node_modules/.bin/jest
- Si Jest globalement
jest
- Avec un script test dans package.json
npm run test
npm test
npm t

```
// package.json
{
  "devDependencies": {
    "jest": "^22.0.6"
  },
  "scripts": {
    "test": "jest"
  }
}
```

```
MacBook-Pro:hello-jest romain$ node_modules/.bin/jest
```

```
PASS __tests__/hello.js
✓ Hello, world ! (3ms)
```

```
Test Suites: 1 passed, 1 total
```

```
Tests: 1 passed, 1 total
```

```
Snapshots: 0 total
```

```
Time: 0.701s, estimated 1s
```

```
Ran all test suites.
```

Tests avec Jest - Watchers



- En mode Watch

```
node_modules/.bin/jest --watchAll
```

```
jest --watchAll
```

```
npm t -- --watchAll
```

```
MacBook-Pro:hello-jest romain$ npm t -- --watchAll
```

```
PASS  __tests__/hello.js
```

```
PASS  __tests__/calc.js
```

```
Test Suites: 2 passed, 2 total
```

```
Tests:      3 passed, 3 total
```

```
Snapshots:  0 total
```

```
Time:       0.65s, estimated 1s
```

```
Ran all test suites.
```

Watch Usage

- > Press f to run only failed tests.
- > Press o to only run tests related to changed files.
- > Press p to filter by a filename regex pattern.
- > Press t to filter by a test name regex pattern.
- > Press q to quit watch mode.
- > Press Enter to trigger a test run.

Tests avec Jest - Coverage



- Avec calcul du coverage

```
node_modules/.bin/jest --coverage
jest --coverage
npm t -- --coverage
```
- Par défaut le coverage s'affiche dans la console et génère des fichiers Clover, JSON et HTML dans le dossier coverage

```
MacBook-Pro:hello-jest romain$ npm t -- --coverage
```

```
PASS  __tests__/calc.js
PASS  __tests__/hello.js
```

```
Test Suites: 2 passed, 2 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.722s, estimated 1s
Ran all test suites.
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
All files	86.67	100	60	100	
calc.js	83.33	100	50	100	
hello.js	100	100	100	100	

Tests avec Jest - Mocks



- Jest intègre par défaut une bibliothèque de Mocks

```
// __tests__/Array.prototype.forEach.js
const names = ['Romain', 'Edouard'];

test('Array forEach method', () => {
  const mockCallback = jest.fn();
  names.forEach(mockCallback);
  expect(mockCallback.mock.calls.length).toBe(2);
  expect(mockCallback).toHaveBeenCalledTimes(2);
});
```

Tests avec Jest - Tester les timers



- La fonction `jest.useFakeTimers()` transforme les timers (`setTimeout`, `setInterval`...) en mock

```
// src/timeout.js
const timeout = (delay, arg) => {
  return new Promise((resolve) => {
    setTimeout(resolve, delay, arg);
  });
};

module.exports = timeout;
```

```
// __tests__/timeout.js
jest.useFakeTimers();

const timeout = require('../src/timeout');

test('waits 1 second', () => {
  const arg = timeout(10000, 'Hello');

  expect(setTimeout).toHaveBeenCalledTimes(1);
  expect(setTimeout).toHaveBeenLastCalledWith(expect.any(Function), 10000,
'Hello');
});
```

Tests avec Jest - React



- Une application créée avec create-react-app est déjà configurée pour fonctionner avec React
- Sinon il faudrait installer des dépendances comme babel, babel-jest...
<https://facebook.github.io/jest/docs/en/tutorial-react.html>

```
// src/App.js
import React, { Component } from 'react';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';

class App extends Component {
  render() {
    return (
      <div>
        <Hello firstName="Romain" />
        <hr />
        <CounterButton/>
      </div>
    );
  }
}

export default App;
```



- Pour tester un composant React il faut en faire le rendu

```
// src/App.test.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
});
```

- 2 inconvénients ici :
 - Nécessite que document existe (exécuter les tests dans un navigateur ou utiliser des implémentations de document côté Node.js comme JSDOM)
 - Les composants enfants du composant testé seront également rendu, le test n'est pas un test unitaire mais un test d'intégration

Tests avec Jest - Snapshot Testing



- Facebook fourni un paquet npm pour simplifier les tests : react-test-renderer
- Ici on fait un simple Snapshot, c'est à dire une capture du rendu du composant, si lors d'un test futur le rendu est modifié le test échoue

```
import React from 'react';
import renderer from 'react-test-renderer';
import { Hello } from './Hello';

test('it renders like last time', () => {
  const tree = renderer
    .create(<Hello />)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```

Tests avec Jest - Shallow Rendering



- On peut également faire appel à `ShallowRenderer` qui ne va faire qu'un seul niveau de rendu, et donc rendre le test unitaire

```
// src/App.test.js
import React from 'react';
import App from './App';
import ShallowRenderer from 'react-test-renderer/shallow';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';

it('renders without crashing', () => {
  const renderer = new ShallowRenderer();
  renderer.render(<App />);
  const result = renderer.getRenderOutput();

  expect(result.type).toBe('div');
  expect(result.props.children).toEqual([
    <Hello firstName="Romain" />,
    <hr />,
    <CounterButton />,
  ]);
});
```

Tests avec Jest - Enzyme



- Facebook recommande également l'utilisation de la bibliothèque Enzyme, créée par AirBnB.
- Elle fournit un API haut niveau (proche de jQuery) pour manipuler les tests des composants

```
import React from 'react';
import { Hello } from './Hello';
import { shallow } from 'enzyme';

test('it renders without crashing with enzyme', () => {
  shallow(<Hello />);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello />);
  expect(wrapper.contains(<div>Hello !</div>)).toEqual(true);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello firstName="Romain"/>);
  expect(wrapper.contains(<div>Hello Romain !</div>)).toEqual(true);
});
```


Tests avec Jest - Tester des événements



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

export class CounterButton extends Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({
      count: this.state.count + 1,
    });
  }

  render() {
    return (
      <button onClick={this.handleClick}>{this.state.count}</button>
    );
  }
}
```

Tests avec Jest - Tester des événements



```
import React from 'react';
import { CounterButton } from './CounterButton';
import { shallow } from 'enzyme';

test('it renders without crashing', () => {
  shallow(<CounterButton />);
});

test('it contains 0 at first rendering', () => {
  const wrapper = shallow(<CounterButton />);
  expect(wrapper.text()).toBe('0');
});

test('it contains 1 after click', () => {
  const wrapper = shallow(<CounterButton />);
  wrapper.simulate('click');
  expect(wrapper.text()).toBe('1');
});
```

Tests avec Jest - Exercices



- Tester unitairement les fonctions liées à Redux :
 - Actions Creators
 - Selectors
 - Reducers
- Tester les composants React (pas les containers) avec Enzyme et les mocks



React Avancé

React Avancé - Higher Order Components



- Permettent d'ajouter des fonctionnalités à un composants de manière générique
- HOC = une fonction qui reçoit un composant en entrée et qui retourne un nouveau composant composé du premier
- Exemple : connect de react-redux, withRouter de react-router-dom

```
▼ <withRouter(Step) number={3} title="Quel est le niveau de l'élève ?">
  ▼ <Route>
    ▼ <Step number={3} title="Quel est le niveau de l'élève ?"> == $r
      ▼ <div className="Step Step3">
        ► <Link className="previous" to="/step2" replace={false}>...</Link>
          <h4 className="title">Quel est le niveau de l'élève ?</h4>
        ► <Choices>...</Choices>
      </div>
    </Step>
  </Route>
</withRouter(Step)>
```

React Avancé - Higher Order Components



▸ Bonne pratique

- Le nom du composant résultant :
`nomDuHOC (NomDuComposant)`
- Les props passées au composant résultant doivent être transmises au composant imbriqué (à l'exception que celle ne servant qu'au HOC) :
`OuterCmp.displayName = `hideable(${InnerCmp.displayName})` ;`

React Avancé - Higher Order Components



▸ Exemple

```
function hideable(InnerComponent) {  
  class OuterComponent extends Component {  
    state = {  
      show: this.props.show,  
    };  
    handleClick = () => {  
      this.setState({  
        show: !this.state.show,  
      });  
    };  
    render() {  
      const {show, ...innerProps} = this.props;  
  
      return (  
        <div className="HideableClock">  
          {this.state.show && <InnerComponent {...innerProps} />}  
          <button onClick={this.handleClick}>  
            {this.state.show ? 'Off' : 'On'}  
          </button>  
        </div>  
      )  
    }  
  }  
  return OuterComponent;  
}
```



- Permettent le rendu dans des éléments DOM distants
- Exemple :
une Modal avec Bootstrap

```
<div class="modal fade" id="exampleModalLong">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLongTitle">Modal title</h5>
        <button type="button" class="close" data-dismiss="modal">
          <span>&times;</span>
        </button>
      </div>
      <div class="modal-body">
        // contenu React à afficher ici...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```




- Le composant Modal pourra faire son rendu dans l'élément

```
class Modal extends Component {
  el = document.createElement('div');

  componentDidMount() {
    modalRoot.appendChild(this.el);
  }

  componentWillUnmount() {
    modalRoot.removeChild(this.el);
  }

  render() {
    return ReactDOM.createPortal(
      this.props.children,
      document.querySelector('#exampleModalLong'),
    );
  }
}
```

```
<Modal>
  Contenu
</Modal>
```



- Context est un objet dont la modification provoque le rendu comme props et state
- Context est utile pour des cas où une valeur doit être fournie globalement pour une hiérarchie de composant (via props il faudrait passer cette valeur à chaque composant ou sous-composant)
- Cas d'utilisation
 - Thèmes
 - Locale
 - Utilisateur connecté
 - Services interchangeables



- Pour créer un Context on utilise la méthode `createContext` de React
Bonne pratique exporter le context (ici 'dark' est la valeur par défaut)

```
import React, { createContext } from 'react';  
export const ThemeContext = createContext('dark');
```

- Pour fournir une nouvelle valeur on utilise le composant `Provider` du context

```
<ThemeContext.Provider value="light">  
  <Navbar />  
</ThemeContext.Provider>
```

- Enfin `Navbar` ou n'importe quel autre composant présent dans la hiérarchie sous `Provider` pourra souscrire aux modifications du context via la propriété `contextType`

```
export class Navbar extends Component {  
  render() {  
    return <div className={"UserList " + this.context}>Menu</div>;  
  }  
}  
  
Navbar.contextType = UserApiContext;
```

React Avancé - Fragments



- React 16 a introduit la possibilité pour un composant de ne plus avoir un élément racine sous forme de tableau

```
function ListItem({ term, definition }) {  
  return [  
    <dt>{term}</dt>,  
    <dd>{definition}</dd>,  
  ];  
}
```

- Ceci pour permettre de créer des composants plus fin dans un contexte où une balise intermédiaire serait problématique

```
function DefinitionList() {  
  const abbrs = { JS: "JavaScript", CSS: "Cascading Style Sheets" };  
  return (  
    <dl>  
      {Object.entries(abbrs).map(([term, definition]) => (  
        <ListItem key={term} term={term} definition={definition} />  
      ))}  
    </dl>  
  );  
}
```



- Les tableaux ont des contraintes :
 - Il faut séparer le contenu par des virgules
 - Il faut ajouter un paramètre key pour la réconciliation
 - Le texte doit être entre guillemets
 - Les commentaires sont différents du JSX

React Avancé - Render Props



- Render Props est une technique consistant à passer une fonction dans les propriétés du composant qui sera en charge du rendu
- Cette fonction aura elle même accès aux propriétés du composants
- Exemple : le composant Field de redux-form

```
const renderField = ({
  input,
  label,
  placeholder,
  type,
  meta: { touched, error },
}) => (
  <div>
    <label>{label}</label>
    <input
      {...input}
      placeholder={placeholder}
      type={type}
      className={classNames({ error: error && touched })}
    />
    {touched && error && (
      <span className={classNames({ error: error && touched })}>{error}</span>
    )}
  </div>
);
```

React Avancé - Render Props



```
<Field  
  name="prenom"  
  type="text"  
  component={renderField}  
  placeholder="Ex: Guillaume"  
  label="Prénom de l'élève"  
/>
```



Redux Avancé

Redux Avancé - Rappels



- Actions
- Actions Creators
- Constants
- Reducers
- Selectors
- `mapStateToProps`
- `dispatch`
- `mapDispatchToProps`



- Redux Form est une bibliothèque qui simplifie la gestion des formulaires en lien avec Redux, notamment la validation des champs
- Installation
npm i redux-form
- Utilisation du reducer de redux-form

```
import { combineReducers } from 'redux';  
import { reducer as reduxFormReducer } from 'redux-form';  
  
export const rootReducer = combineReducers({  
  // ...  
  form: reduxFormReducer,  
});
```



▸ Exemple

```
const ContactForm = (props) => {
  const { invalid, submit } = props;
  return (
    <div>
      <Field
        name="prenom"
        type="text"
        component={renderField}
        label="Prénom de l'élève"
      />
      <Field
        name="nom"
        type="text"
        component={renderField}
        label="Nom de l'élève"
      />
      <NextButton disabled={invalid} onClick={submit}>Envoyer</NextButton>
    </div>
  );
};

export const Form = reduxForm({
  form: 'contact',
  validate,
  destroyOnUnmount: false,
})(ContactForm);
```

Redux Avancé - Redux Persist



- Permet de faire persister le state entre 2 démarrages de l'application
- Le state sera automatiquement stocké dans le localStorage, sessionStorage...
- Documentation
<https://github.com/rt2zz/redux-persist>

```
import { persistStore, persistReducer } from 'redux-persist';
import storage from 'redux-persist/lib/storage';

import rootReducer from './reducers';

const persistConfig = {
  key: 'root',
  storage,
};

const persistedReducer = persistReducer(persistConfig, rootReducer);
```

Redux Avancé - Redux Persist



- Le composant PersistGate permet de passer la version précédente du state à l'application

```
import { PersistGate } from 'redux-persist/integration/react'

function App() {
  return (
    <Provider store={store}>
      <PersistGate loading={null} persistor={persistor}>
        <RootComponent />
      </PersistGate>
    </Provider>
  );
};
```

Redux Avancé - Middleware



- Un middleware est un plugin qui est exécuté à chaque dispatch. Il permet d'accéder au store et à l'action
- Pour passer au prochain middleware on utilise la fonction next comme ci-dessous

```
import { createStore, combineReducers, applyMiddleware } from "redux";
import { composeWithDevTools } from "redux-devtools-extension";

const rootReducer = combineReducers({
  // ...
});

const logger = store => next => action => {
  console.group(action.type);
  console.info("dispatching", action);
  let result = next(action);
  console.log("next state", store.getState());
  console.groupEnd();
  return result;
};

const store = createStore(
  rootReducer,
  composeWithDevTools(applyMiddleware(logger))
);
```



- react-persist
 - Ajouter react-persist pour faire persister le state dans le localStorage
<https://github.com/rt2zz/redux-persist#basic-usage>
- redux-form
 - En prenant exemple sur <https://redux-form.com/8.2.1/examples/syncvalidation/>
 - Créer un formulaire avec des champs : name, email et phone
 - Ecrire une fonction de validation
 - Ecrire un fonction de rendu pour chaque élément (name type="text", email type="email", phone type="phone")
- redux-thunk
 - Au submit du formulaire, faire un dispatch d'une action userCreateRequested, cette action fera :
 - dispatch de userCreate
 - La requête POST <https://jsonplaceholder.typicode.com/users>
 - dispatch de userCreateSuccess (facultatif userCreateError)



Optimisation des performances

Optimisation - Introduction



- Plus l'application React va grandir, plus le nombre de composants va être élevé et donc les appels à render long à exécuter
- Les composants les plus problématiques : ceux recevant en paramètre une liste d'élément dont chacun sera rendu sous forme d'un composant. Un changement dans une liste de 1000 éléments == 1000 appels à render + 1000 mise à jour du DOM potentielle
- Les performances des différents frameworks JavaScript sur des listes : <https://github.com/krausest/js-framework-benchmark>

Optimisation - Keys



- Au moment de la réconciliation, React va comparer la version précédente du Virtual DOM d'un composant avec la version actuelle (juste après l'appel à render)
- Avec un tableau passé en props ou state, si une valeur est insérée au début, l'ensemble des éléments du DOM devront être mis à jour.
- Pour éviter cela, il faut passer au Virtual DOM un paramètre key dans les props lui permettant d'établir un lien entre l'élément du Virtual DOM et l'élément du DOM
- Choisir une valeur unique, et non modifiée en cas de mise à jour de l'élément (id de la database, uuid généré à la création de l'élément)

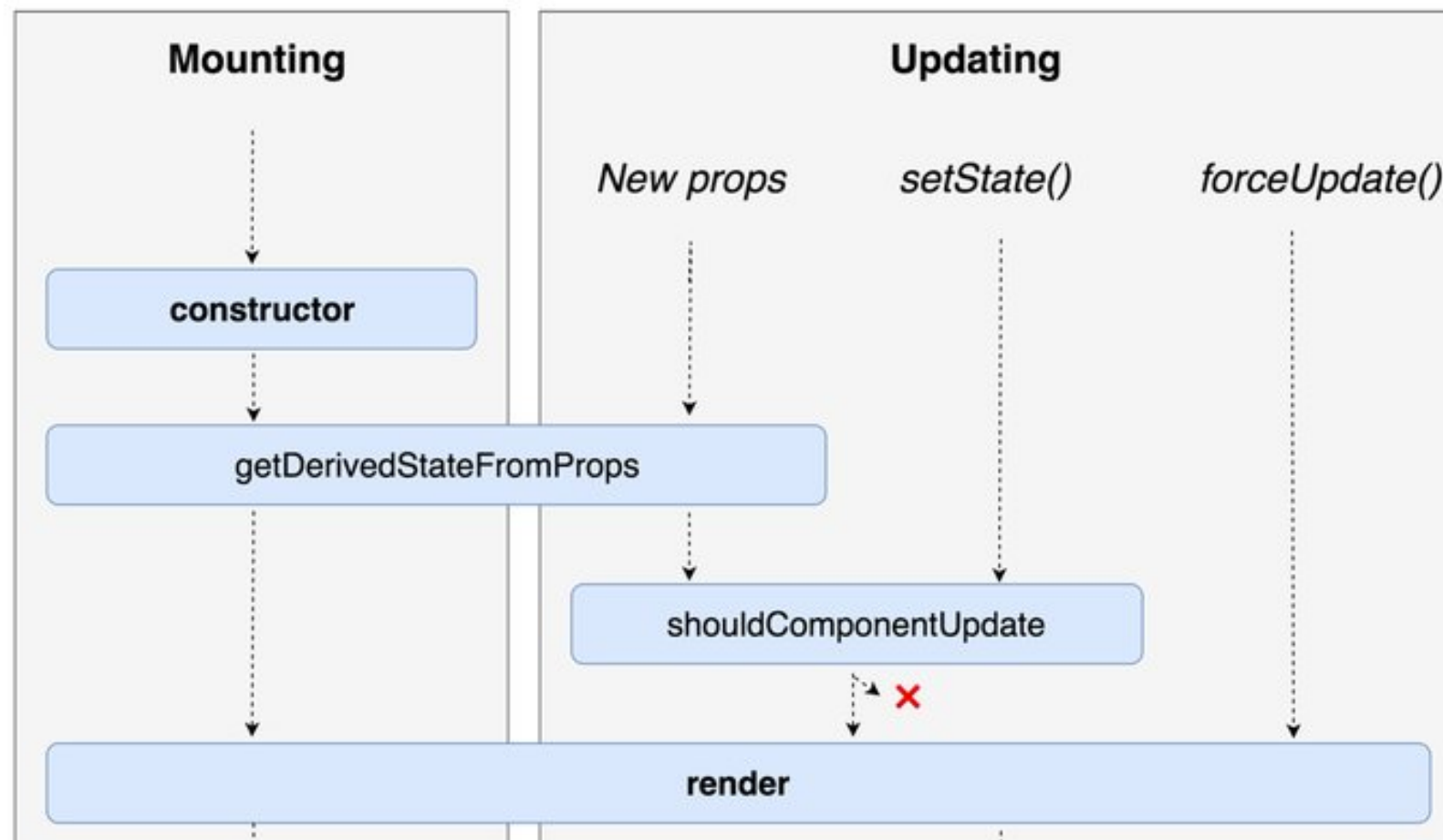
```
function DefinitionList() {  
  const abbrs = { JS: "JavaScript", CSS: "Cascading Style Sheets" };  
  return (  
    <dl>  
      {Object.entries(abbrs).map(([term, definition]) => (  
        <ListItem key={term} term={term} definition={definition} />  
      ))}  
    </dl>  
  );  
}
```

Optimisation - shouldComponentUpdate



- Lorsque que le state ou les props d'un éléments sont mis à jour, une cascade de render va s'effectuer pour les composants enfants
- Il est possible de bloquer les render liés à l'update d'un élément en créant une méthode `shouldComponentUpdate` sur l'élément

```
shouldComponentUpdate(nextProps) {  
  return this.props.todos !== nextProps.todos;  
}
```



Optimisation - PureComponent



- Un composant "pur" est un composant contenant une méthode `shouldComponentUpdate` vérifiant que chacune des propriétés est différente de la valeur précédente
- Lorsque qu'on utilise la classe `PureComponent`, il faudra donc mettre à jour les tableaux et les objets de façon "immuable"

```
class TodoList extends PureComponent {
  render() {
    const todoItems = this.props.todos.map((todo) => (
      <TodoItem key={todo.id} todo={todo}
        onDelete={() => this.props.onDelete(todo)} />
    ));

    return (
      <div className="TodoList">
        {todoItems}
      </div>
    );
  }
}
```

Optimisation - Exercice



- Transformer TodoForm, TodoList et TodoItem en PureComponent



Immuabilité

Immuabilité - Introduction



- Lors de la modification d'un objet, le changement peut-être muable en modifiant l'objet d'origine ou immuable en créant un nouvel objet
- Les algorithmes de détections de changements préféreront les changements immuables, ayant ainsi juste à comparer les références plutôt que l'ensemble du contenu de l'objet
- Exemple, en JS les tableaux sont muables, les chaines de caractères immuables

```
const firstName = 'Romain';  
firstName.concat('Edouard');  
console.log(firstName); // Romain  
  
const firstNames = ['Romain'];  
firstNames.push('Edouard');  
console.log(firstNames.join(', ')); // Romain, Edouard
```



▸ Ajouter à la fin

```
const firstNames = ['Romain', 'Edouard'];

function append(array, value) {
  return [...array, value];
}

const newfirstNames = append(firstNames, 'Jean');
console.log(newfirstNames.join(', ')); // Romain, Edouard, Jean
console.log(firstNames === newfirstNames); // false
```

▸ Ajouter au début

```
const firstNames = ['Romain', 'Edouard'];

function prepend(array, value) {
  return [value, ...array];
}

const newfirstNames = prepend(firstNames, 'Jean');
console.log(newfirstNames.join(', ')); // Jean, Romain, Edouard
console.log(firstNames === newfirstNames); // false
```




- Ajouter à un indice donné

```
const firstNames = ['Romain', 'Edouard'];

function insertAt(array, value, i) {
  return [
    ...array.slice(0, i),
    value,
    ...array.slice(i),
  ];
}

const newfirstNames = insertAt(firstNames, 'Jean', 1);
console.log(newfirstNames.join(', ')); // Romain, Jean, Edouard
console.log(firstNames === newfirstNames); // false
```



▸ Modifier un élément

```
const firstNames = ['Romain', 'Edouard'];

function modify(array, value, i) {
  return [
    ...array.slice(0, i),
    value,
    ...array.slice(i + 1),
  ];
}

const newfirstNames = modify(firstNames, 'Jean', 1);
console.log(newfirstNames.join(', ')); // Romain, Jean
console.log(firstNames === newfirstNames); // false
```



- Supprimer un élément

```
const firstNames = ['Romain', 'Edouard'];

function remove(array, i) {
  return [
    ...array.slice(0, i),
    ...array.slice(i + 1),
  ];
}

const newfirstNames = remove(firstNames, 1);
console.log(newfirstNames.join(', ')); // Romain
console.log(firstNames === newfirstNames); // false
```



- Ajouter un élément

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

function add(object, key, value) {
  return {
    ...object,
    [key]: value,
  };
}

const newContact = add(contact, 'city', 'Paris');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain","lastName":"Bohdanowicz","city":"Paris"}
console.log(contact === newContact); // false
```



▸ Modifier un élément

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

function modify(object, key, value) {
  return {
    ...object,
    [key]: value,
  };
}

const newContact = modify(contact, 'firstName', 'Thomas');
console.log(JSON.stringify(newContact));
// {"firstName":"Thomas","lastName":"Bohdanowicz"}
console.log(contact === newContact); // false
```



- Supprimer un élément

```
const contact = {
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
};

function remove(object, key) {
  const { [key]: val, ...rest } = object;
  return rest;
}

const newContact = remove(contact, 'lastName');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain"}
console.log(contact === newContact); // false
```

Immuabilité - Immutable.js



- Pour simplifier la manipulation d'objets ou de tableaux immuables, Facebook a créé Immutable.js
- Installation
`npm install immutable`

Immuabilité - Immutable.js List



▸ Ajouter à la fin

```
const immutable = require('immutable');  
  
const firstNames = immutable.List(['Romain', 'Edouard']);  
  
const newfirstNames = firstNames.push('Jean');  
console.log(newfirstNames.join(', ')); // Romain, Edouard, Jean  
console.log(firstNames === newfirstNames); // false
```

▸ Ajouter au début

```
const immutable = require('immutable');  
  
const firstNames = immutable.List(['Romain', 'Edouard']);  
  
const newfirstNames = firstNames.unshift('Jean');  
console.log(newfirstNames.join(', ')); // Jean, Romain, Edouard  
console.log(firstNames === newfirstNames); // false
```


Immuabilité - Immutable.js List



- Ajouter à un indice donné

```
const immutable = require('immutable');  
  
const firstNames = immutable.List(['Romain', 'Edouard']);  
  
const newfirstNames = firstNames.insert(1, 'Jean');  
console.log(newfirstNames.join(', ')); // Romain, Jean, Edouard  
console.log(firstNames === newfirstNames); // false
```

Immuabilité - Immutable.js List



▸ Modifier un élément

```
const immutable = require('immutable');  
  
const firstNames = immutable.List(['Romain', 'Edouard']);  
  
const newfirstNames = firstNames.set(1, 'Jean');  
console.log(newfirstNames.join(', ')); // Romain, Jean  
console.log(firstNames === newfirstNames); // false
```

Immuabilité - Immutable.js List



- Supprimer un élément

```
const immutable = require('immutable');  
  
const firstNames = immutable.List(['Romain', 'Edouard']);  
  
const newfirstNames = firstNames.delete(1);  
console.log(newfirstNames.join(', ')); // Romain  
console.log(firstNames === newfirstNames); // false
```

Immuabilité - Immutable.js Map



- Ajouter un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
});

const newContact = contact.set('city', 'Paris');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain","lastName":"Bohdanowicz","city":"Paris"}
console.log(contact === newContact); // false
```

Immuabilité - Immutable.js Map



▸ Modifier un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
});

const newContact = contact.set('firstName', 'Thomas');
console.log(JSON.stringify(newContact));
// {"firstName":"Thomas","lastName":"Bohdanowicz"}
console.log(contact === newContact); // false
```

Immuabilité - Immutable.js Map



- Supprimer un élément

```
const immutable = require('immutable');

const contact = immutable.Map({
  firstName: 'Romain',
  lastName: 'Bohdanowicz',
});

const newContact = contact.remove('lastName');
console.log(JSON.stringify(newContact));
// {"firstName":"Romain"}
console.log(contact === newContact); // false
```

Immuabilité - Exercice



- Installer Immutable.js
- Utiliser Immutable.js pour manipuler le tableau dans todosReducer
- Aide : <https://redux.js.org/recipes/using-immutablejs-with-redux>