

TP's  
***React.js Module***

ENI, Nicolas Hodicq

Version 1.0, 2018-05-16

# Table des matières

1. Installation du poste de développement .....	1
1.1. Node.js .....	1
1.1.1. Yarn .....	1
1.2. Devtools .....	2
1.3. IDE .....	3
1.3.1. Visual Studio Code .....	3
1.3.2. Webstorm .....	3
2. Premières applications React .....	4
2.1. Via CDN .....	4
2.2. Via create-react-app .....	8
2.2.1. Commandes .....	8
2.2.2. Architecture et dépendances .....	9
3. Composants : Bases .....	12
3.1. JSX .....	12
3.1.1. Objects .....	12
3.1.2. Expressions .....	13
3.1.3. Attributs .....	13
3.2. Composants .....	15
3.2.1. Déclarer un composant .....	15
3.2.2. Utiliser un composant .....	16
3.2.3. TP .....	16
3.2.4. Props .....	17
3.2.5. TP .....	18
3.2.6. State .....	19
3.2.7. TP .....	20
3.3. Lifecycle hooks .....	21
3.3.1. TP .....	22
3.4. Handling events .....	23
3.4.1. TP .....	23
3.5. Conditionnal .....	24
3.5.1. Elements variables .....	25
3.5.2. Inline If with Logical && Operator .....	26
3.5.3. Inline If-Else with Conditional Operator .....	26
3.5.4. Preventing Component from Rendering .....	27
3.5.5. TP .....	27
4. Composants : Avancés .....	29
4.1. Listes .....	29

4.2. Forms .....	31
4.2.1. TP .....	34
4.3. Composition .....	35
4.4. PropTypes .....	37
4.5. TP : Todo Application .....	39
5. API .....	40
5.1. Context .....	40
5.1.1. TP .....	42
5.2. Ref - Forwarding ref .....	44
5.2.1. Ref .....	44
5.2.2. Forwarding Ref .....	46
6. React Router .....	47
6.1. Installation .....	47
6.2. Routes .....	49
6.2.1. Router .....	49
6.2.2. Routes .....	50
6.3. Switch .....	54
6.4. Link .....	55
6.4.1. Link .....	55
6.4.2. NavLink .....	55
6.5. TP .....	56
7. Redux .....	58
7.1. Actions synchrones .....	58
7.2. Actions asynchrones avec Redux-thunk .....	62
7.3. TP .....	68
8. Code quality .....	69
8.1. Eslint .....	69
8.1.1. react-scripts .....	69
8.1.2. Airbnb rules .....	71
8.2. Jest .....	72
8.2.1. Snapshot testing .....	73
8.3. Enzyme .....	75
9. Annexes .....	79

# Chapitre 1. Installation du poste de développement

## 1.1. Node.js

Node.js permet d'exécuter des fichiers JavaScript sur une machine hôte. Ce n'est pas un prérequis pour développer une application React.js, nous verrons qu'il est possible de s'en passer. Dans les faits, tous les outils modernes de développement web, et React n'échappe pas à la règle, reposent sur Node.js

Node.js doit donc être installé sur le poste de développement.

- Vérifiez que Node.js est installé

```
node -v
```

- Si ce n'est pas le cas, installez Node.js depuis le [site officiel](#)

### 1.1.1. Yarn

[Yarn](#) est un outil de gestion de dépendances Node.js qui a été open sourcé par Facebook. Il se veut plus rapide que NPM, avec une meilleure gestion des dépendances transitives et un meilleur cache.

C'est de fait l'outil de référence pour un projet React. Npm fonctionne très bien aussi, mais si vous avez yarn et npm installés sur votre machine, yarn sera toujours pris en priorité par les utilitaires React.

```
// Yarn est installé sur votre machine  
yarn help
```

## 1.2. Devtools

En tant que développeur web, nous utilisons quotidiennement les outils de développement fournis par les navigateurs. Ces outils ne sont pas complètement optimisés pour des applications React.js

Facebook et la communauté React met à disposition des extensions pour les navigateurs afin de rendre la vie du développeur plus aisée ;)

- Installez l'extension en fonction de votre navigateur préféré
  - Chrome : [React Developer Tools](#)
  - Firefox : [React Developer Tools](#)



### React Developer Tools

proposé par [Facebook](#)

★★★★★ (943)

[Outils de développement](#)

1 007 203 utilisateurs

AJOUTÉ À CHROME

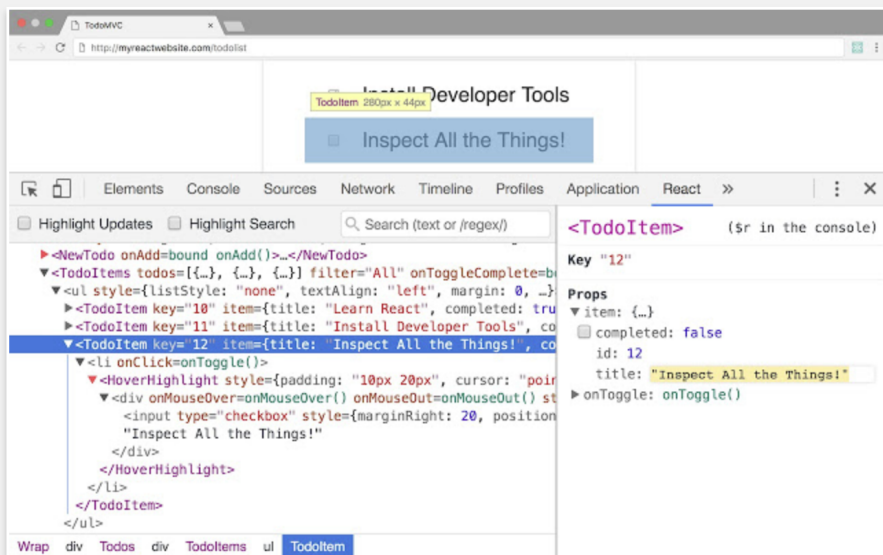


PRÉSENTATION

AVIS

ASSISTANCE

ARTICLES SIMILAIRES



Compatible avec votre appareil

Adds React debugging tools to the Chrome Developer Tools.

React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows you to inspect the React component hierarchies in the Chrome Developer Tools.

You will get a new tab called React in your Chrome DevTools. This shows you the root React components that were rendered on the page, as well as the subcomponents that they ended up rendering.

By selecting one of the components in the

[Site Web](#)

[Signaler un abus](#)

Informations supplémentaires

Version : 3.2.1

Mise à jour : 24 mars 2018

Taille : 335KIB

Langue : English

## 1.3. IDE

L'IDE est un élément essentiel pour une expérience de développement optimisée. Vous pouvez bien sûr utiliser un bon vieux éditeur de texte ou bien même Vim ;) Il existe aujourd'hui toutefois des outils pour nous faciliter la tâche. Vous trouverez sur le poste de développement deux IDE :


### 1.3.1. Visual Studio Code

IDE open source développé par Microsoft. C'est un des leader actuellement, il nécessite toutefois une configuration via des add-ons.

### 1.3.2. Webstorm

IDE développé par JetBrains et payant. De mon point de vue, l'IDE le plus performant pour développer des applications web. Il existe une version payante, une version d'essai (limitée dans le temps) et une version EAP (Early Access Program). La version EAP est celle installée sur votre poste.

Pour ceux qui choisirait cet IDE, il existe une extension pour Chrome qui permet de debugger directement dans l'IDE ([ici](#))



# JetBrains IDE Support

proposé par [www.jetbrains.com](http://www.jetbrains.com)

★★★★★ (690) | [Outils de développement](#) | 391 168 utilisateurs


[AJOUTÉ À CHROME](#)

PRÉSENTATION

AVIS

ASSISTANCE

ARTICLES SIMILAIRES



JetBrains IDE Support Chrome Extension

## WebStorm

# JetBrains Chrome Extension

Paul Everitt  
Developer Advocate at JetBrains

© JetBrains. All rights reserved

Compatible avec votre appareil

HTML/CSS/JavaScript editing and JavaScript debugging using JetBrains IDEs.

With the JetBrains IDE Support extension for Google Chrome you can debug JavaScript code in Chrome from WebStorm, PhpStorm, IntelliJ IDEA Ultimate, PyCharm Professional, and RubyMine. In addition to that, you can see the changes you make in HTML or CSS files in the browser right away, without reloading the page, thanks to the Live Edit plugin.

[Site Web](#)

[Signaler un abus](#)

**Informations supplémentaires**

Version : 2.0.9

Mise à jour : 7 juillet 2016

Taille : 126KIB

Langue : English

# Chapitre 2. Premières applications React

## 2.1. Via CDN

React.js est une librairie JavaScript. Il est possible de créer une page HTML qui référence React en tant qu'une librairie tierce via un import JavaScript classique.

- Créez une page HTML et réalisez les imports de React

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>React va CDN</title>
  <script crossorigin src=
"https://unpkg.com/react@16/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
</head>
<body>
</body>
</html>
```

- Ajoutez le code comme dans l'exemple suivant

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>React va CDN</title>
  <script crossorigin src=
"https://unpkg.com/react@16/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
</head>
<body>
  <!-- Ajout d'un conteneur qui va recevoir les composants managés par React.js -->
  <div id="react-root"></div>

  <script>
    // Création d'un composant React
    var element = React.createElement('div', { className: 'whatever' }, 'First
component');

    // Ajout du composant React dans le conteneur possédant l'id react-root
    ReactDOM.render(
      element,
      document.getElementById('react-root'),
    );
  </script>
</body>
</html>
```



Vous pouvez remarquer que le code JavaScript est écrit en ES5. Tous les navigateurs ne supportent pas encore les imports ES6, ni toutes les API.



[Babel.js](#) est un JavaScript compiler qui permet d'utiliser toutes les features des nouvelles versions de JavaScript avant qu'elles soient implémenter dans les navigateurs. Tous les framework JS actuels utilisent d'une façon ou d'une autre cet outil ou ce type d'outil. Babel est aujourd'hui le plus populaire.

Il est possible d'importer Babel dans la page afin de pouvoir utiliser les arrow function, les class, let, const, .... mais aussi la syntaxe JSX (nous reviendrons sur cette notion plus loin).

- Ajoutez le code suivant dans votre page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>React va CDN</title>
  <script crossorigin src=
"https://unpkg.com/react@16/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
  <script crossorigin src="https://cdnjs.cloudflare.com/ajax/libs/babel-
standalone/6.26.0/babel.min.js"></script>
</head>
<body>
  <div id="react-root"></div>

  <script type="text/babel">
    class FirstComponent extends React.Component{
      render(){
        return (
          <div className="whatever">
            First component
          </div>
        );
      }
    }

    ReactDOM.render(
      <FirstComponent/>,
      document.getElementById('react-root'),
    );
  </script>
</body>
</html>
```



La création d'une application React peut se faire aussi simplement que ce que nous venons de voir. Cependant, l'outillage que nous utilisons aujourd'hui est plus complexe et complet. Cette approche pour créer une application React est très marginale.

## 2.2. Via create-react-app

Facebook et la communauté open source mettent à disposition un module Node.js qui permet de créer une application React avec tout l'outillage nécessaire. Le squelette de l'application créée sera le plus simple possible mais permettra aussi d'avoir une base commune pour des configurations plus avancées.

- Installez le module `create-react-app`

```
npm install -g create-react-app
```

- Créez votre application

```
create-react-app my-app  
cd my-app
```

### 2.2.1. Commandes

- Démarrez l'application

```
yarn start
```



L'application vient de démarrer, votre navigateur par défaut a ouvert une page qui pointe sur l'URL localhost:3000. Un serveur de développement est mis en place, à chaque fois que vous modifierez un fichier dans vos sources, le serveur mettra l'application à jour dans votre navigateur. C'est le **live reload**.

- Stoppez le process dans votre terminal
- Démarrez les tests

```
yarn test
```



Les tests unitaires de l'application sont joués. A chaque changement dans un fichier les tests seront rejoués.

- Stoppez le process dans votre terminal
- Faites un build de release de l'application

```
yarn build
```



Le dossier build vient d'être créé et prêt à être déployé sur un un serveur web

### 2.2.2. Architecture et dépendances

Le projet est un projet Node avec un package.json qui définit les dépendances et les lanceurs

```
{
  "name": "my-app",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^16.3.2",
    "react-dom": "^16.3.2",
    "react-scripts": "1.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  }
}
```

Les dépendances sont :

- react
- react-dom
- react-scripts

Toutes ses dépendances proviennent du repository github de React et sont maintenues par Facebook et la communauté open source.

L'architecture du projet est définie comme suit :

- Dossier **public** ⇒ contient la page html d'entrée de l'application web avec le conteneur cible pour l'application React

Sources: 1. my-app/public/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="theme-color" content="#000000">
    <!--
      manifest.json provides metadata used when your web app is added to the
      homescreen on Android. See
      https://developers.google.com/web/fundamentals/engage-and-retain/web-app-manifest/
    -->
    <link rel="manifest" href="%PUBLIC_URL%/manifest.json">
    <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the 'public' folder during the build.
      Only files inside the 'public' folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running 'npm run build'.
    -->
    <title>React App</title>
  </head>
  <body>
    <noscript>
      You need to enable JavaScript to run this app.
    </noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run 'npm start' or 'yarn start'.
      To create a production bundle, use 'npm run build' or 'yarn build'.
    -->
  </body>
</html>
```

- Dossier `src` ⇒ contient l'application React

Sources: 2. my-app/src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

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



Vous pouvez remarquer que chaque composant (ex: App) est défini par 3 fichiers (js, css, test.js).



D'un point de vue CSS, la méthodologie préconisée (cf l'exemple App.css) est la méthodologie [BEM](#)

# Chapitre 3. Composants : Bases

## 3.1. JSX

La syntaxe JSX (JavaScript Syntax eXtension) a été créée par Facebook pour React. Cela va nous permettre d'écrire une syntaxe XML dans du JavaScript. Ecrire du code HTML dans un fichier JS n'est pas nouveau, mais nous l'écrivons habituellement sous forme de chaîne de caractère avec tous les inconvénients que nous connaissons. Pas d'auto-complétion, pas de recherche de référence, ... Bref, ce n'est pas pratique pour notre confort de développeurs, la maintenance d'une application, les non régressions.

Nous avons appris depuis longtemps que la bonne pratique est la séparation des responsabilités, le design dans la couche CSS, la structure de données dans l'HTML, et tout le code dynamique dans le JS. Facebook, via React, casse cette approche en prenant le parti qu'il y a un couplage fort entre la structure de données et le code dynamique. Cela a donc du sens à ce que ce code soit "au même endroit".

Voici un exemple :

```
const element = <h1>Hello, world!</h1>;
```



Un élément React qui contient un élément du DOM qui a des enfants (écriture sur plusieurs lignes) devra être entouré de parenthèses.

```
const element = (  
  <h1 className="greeting">  
    Hello, world!  
  </h1>  
)
```

### 3.1.1. Objects

Concrètement, un élément écrit en JSX peut être écrit en JS. L'exemple ci dessus pourrait être écrit comme cela :

```
const element = React.createElement(  
  'h1',  
  {className: 'greeting'},  
  'Hello, world!'  
)
```

Dans les deux cas, React transforme l'élément en objet :

```
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

### 3.1.2. Expressions

Le JSX ne permet pas juste de pouvoir écrire du contenu HTML dans un fichier JS. Il tire aussi toute la puissance du langage JavaScript. Ainsi il est possible d'évaluer des expressions JS dans un élément.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Harper',
  lastName: 'Perez'
};

const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

### 3.1.3. Attributs

Les expressions peuvent aussi être utilisées pour définir dynamiquement la valeur d'un attribut d'une balise XML.

```
// Chaîne de caractère
const element = <div tabIndex="0"></div>;

// Evaluation d'une expression
const element = <img src={user.avatarUrl}></img>;
```





Issue de la documentation ⇒ Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names. For example, class becomes className in JSX, and tabIndex becomes tabIndex.

## 3.2. Composants

### 3.2.1. Déclarer un composant

React permet d'écrire un composant de deux façons différentes.

#### Function

C'est la syntaxe la plus simple pour définir un composant.

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const element = <Welcome name="Sara" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

#### Class

Il est également possible de définir un composant via une class ES6

```
class Welcome extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}</h1>;  
  }  
}  
  
const element = <Welcome name="Sara" />;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
);
```

#### Choisir la bonne syntaxe

La bonne pratique est d'utiliser la déclaration d'un composant via une fonction si le composant ne définit pas de state ou de lifecycle hooks. Nous verrons aussi que les styles checker (Eslint) lèveront des warnings pour nous aider à avoir un code homogène entre les différents développeurs d'un projet.

### 3.2.2. Utiliser un composant

Un composant créé peut être utilisé comme n'importe quel élément du DOM :

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
function App() {  
  return (  
    <div>  
      <Welcome name="Sara" />  
      <Welcome name="Cahal" />  
      <Welcome name="Edite" />  
    </div>  
  );  
}  
  
ReactDOM.render(  
  <App />,  
  document.getElementById('root')  
);
```

### 3.2.3. TP

- Reprenez le projet `my-app`
- Le composant `App.js` ne possède pas de `state` ni de lifecycle hooks.

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

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>
    );
  }
}

export default App;
```

- Ecrivez le sous forme d'une fonction.

### 3.2.4. Props

Les props sont le moyen de passer des données d'un composant parent à un composant enfant. Le type de données peut être n'importe quel type de données JavaScript (primitif et complexe).



Les props sont READ ONLY.

Reprenons l'exemple précédent :

```
// Composant de type fonction
// Un objet props est passé en argument de la fonction
// Les propriétés de cet objet sont les attributs XML déclarés lorsque le
composant est instancié
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

// Composant de type Class
// Une propriété props est définie dans l'objet (accessible via this.props)
// Les propriétés de cet objet sont les attributs XML déclarés lorsque le
composant est instancié
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}

function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
    </div>
  );
}

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



Il est impossible de transmettre des données d'un composant enfant vers le parent, mais il est possible de passer une fonction en tant que props, cela ressemble fortement aux callback fortement utilisés en JavaScript. Cette approche permet de communiquer avec le composant parent.

### 3.2.5. TP

- Créez une application React qui affiche une liste de formation

## Formations :

React.js

React Native

Angular

Typescript



La bonne approche est de découper l'interface en petits composants simples. Pour l'exemple ci dessus, une piste pourrait être, TrainingList, TrainingItem, TrainingTitle

### 3.2.6. State

Un composant doit parfois garder un état qui peut évoluer au fil du temps. La notion de state répond à ce besoin contrairement aux props qui pour rappel sont **read only**.

```

class Clock extends React.Component {
  constructor(props) {
    super(props);

    // Création du state
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <!-- Utilisation de la valeur de la propriété date du state -->
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

```

Il est désormais possible de modifier la valeur du state

```

// de manière synchrone
this.setState({
  date: new Date()
});

// de manière asynchrone
this.setState((prevState, props) => ({
  counter: prevState.counter + props.increment
}));

```



Lorsque le state d'un composant est modifié, le composant est alors redessiné ⇒ la méthode render est à nouveau exécutée.

### 3.2.7. TP

- Dans l'application que vous avez créée précédemment, ajoutez un label qui affiche 'OFFLINE' ou 'ONLINE' en fonction de la connectivité.



Allez au plus simple en ajoutant un listener sur les événements online et offline `window.addEventListener("offline", () => console.log(navigator.onLine), false);`

### 3.3. Lifecycle hooks

Un composant React possède un cycle de vie. Dans un composant qui étend `React.Component`, React met à disposition des **lifecycle methods** que l'on peut surcharger pour effectuer des traitements à différents moments du cycle de vie. Les méthodes préfixées par **will** sont appelées avant une étape du cycle. Celles préfixées par **did** sont elles appelées après ;)



N'hésitez pas à vous diriger vers la [documentation officielle](#) qui répertorie toutes les étapes du cycle de vie.

La version 16.3 de React.js déprécie certains lifecycles et en ajoute 2 nouveaux. Les lifecycles mentionnés avec UNSAFE sont amenés à disparaître, par exemple : `componentWillMount()` / `UNSAFE_componentWillMount()`. A date, il est important pour des développeurs se formant à React.js de comprendre rapidement cette problématique. En effet, il y a de forte chance que si vous intervenez sur un projet React.js avec un peu d'historique, vous y soyez confronté.



Je vous conseille de prendre le temps de lire ce post du [blog](#) React.js. Si vous souhaitez approfondir le sujet, la [video](#) de Dan Abramov.



Pour les TP's suivants nous nous baserons sur les nouveaux lifecycles.

Dans l'exemple suivant vous pouvez remarquer que deux lifecycles methods ont été déclarées dans le composant :

- `componentDidMount` ⇒ Une des lifecycles methods les plus utilisées. C'est le bon moment si vous avez le besoin de fetch des données.
- `componentWillUnmount` ⇒ Il est important de bien nettoyer les listeners ou autres pour des questions d'optimisation de l'application



```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```



Lorsqu'un composant React est rendu, ses enfants le sont aussi. Pour des questions de performances, le lifecycle `shouldComponentUpdate` permet de définir si le composant doit être rendu ou non. Dans des applications possédant une base de code importante, il est souvent nécessaire d'optimiser le nombre de composants rendus.

### 3.3.1. TP

- Reprenez l'application avec la liste de formation.
- Affichez dans la console la valeur de la propriété `name` du composant `TrainingItem` lorsque le composant vient d'être monté.
- Loguez aussi le lifecycle `shouldComponentUpdate` pour tous vos composants

## 3.4. Handling events

Nos traitements doivent souvent réagir en fonction d'évènements, que ce soit des interactions de l'utilisateur sur l'interface ou bien les évènements issus du DOM.

React permet d'écouter ces évènements comme par exemple le fameux onclick :

```
<button onClick={activateLasers}>  
  Activate Lasers  
</button>
```



Les évènements sur les composants React sont toujours en camelCase contrairement aux évènements du DOM qui sont lowercase.

### 3.4.1. TP

- Reprenez l'application avec la liste de formation.
- Au click sur le composant TrainingItem, ouvrez une popup (alert) qui affiche le nom de la formation qui a été cliquée
- Affichez dans la console le type de l'évènement (event.nativeEvent.type)

## 3.5. Conditionnal

Pour l'instant, toutes les interfaces que vous avez manipulées affichaient toujours l'ensemble des composants. Cependant dans la vraie vie d'une application, cela n'arrive que très rarement. Nous avons donc le besoin de pouvoir conditionner le rendu d'un composant en fonction de conditions.

Dans la documentation de React, les techniques suivantes sont décrites :



Toutes les techniques sont possibles grâce à la syntaxe JSX qui étend le langage JS. Ce sont des possibilités que nous offre déjà le JS, c'est donc assez naturel.

### 3.5.1. Elements variables

```
class LoginControl extends React.Component {
  constructor(props) {
    super(props);
    this.handleLoginClick = this.handleLoginClick.bind(this);
    this.handleLogoutClick = this.handleLogoutClick.bind(this);
    this.state = {isLoggedIn: false};
  }

  handleLoginClick() {
    this.setState({isLoggedIn: true});
  }

  handleLogoutClick() {
    this.setState({isLoggedIn: false});
  }

  render() {
    const isLoggedIn = this.state.isLoggedIn;

    // Expression ternaire, retourne un composant ou l'autre en fonction de la
    condition
    const button = isLoggedIn ? (
      <LogoutButton onClick={this.handleLogoutClick} />
    ) : (
      <LoginButton onClick={this.handleLoginClick} />
    );

    return (
      <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {button}
      </div>
    );
  }
}
```

### 3.5.2. Inline If with Logical && Operator

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      <h1>Hello!</h1>

      // Evaluation de la condition directement dans le block JSX
      // Cette évaluation vient du JS qui permet d'écrire 'condition &&
      console.log('condition is true')
      {unreadMessages.length > 0 &&
        <h2>
          You have {unreadMessages.length} unread messages.
        </h2>
      }
    </div>
  );
}
```

### 3.5.3. Inline If-Else with Conditional Operator

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      // Evaluation d'un expression ternaire dans le block JSX
      {isLoggedIn ? (
        <LogoutButton onClick={this.handleLogoutClick} />
      ) : (
        <LoginButton onClick={this.handleLoginClick} />
      )}
    </div>
  );
}
```

### 3.5.4. Preventing Component from Rendering

Lorsque l'on ne veut pas rendre un composant dans l'arbre du DOM, il faut explicitement retourner null. Le composant WarningBanner sera présent quand la condition `!props.warn` est fausse.

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Warning!
    </div>
  );
}

class Page extends React.Component {
  constructor(props) {
    super(props);
    this.state = {showWarning: true};
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(prevState => ({
      showWarning: !prevState.showWarning
    }));
  }

  render() {
    return (
      <div>
        <WarningBanner warn={this.state.showWarning} />
        <button onClick={this.handleClick}>
          {this.state.showWarning ? 'Hide' : 'Show'}
        </button>
      </div>
    );
  }
}
```

### 3.5.5. TP

- Reprenez l'application avec la liste de formation.
- Faites évoluer le composant TrainingItem pour qu'il n'affiche pas les item dont le nom est `angular` ;)

- Affichez le label 'OFFLINE'/'ONLINE' uniquement quand il n'y pas de connectivité

# Chapitre 4. Composants : Avancés

## 4.1. Listes

Dans une application web, nous sommes souvent amenés à gérer des données sous formes de listes. Contrairement à d'autres frameworks web tel Angular, React ne dispose de composants ou d'API particulière pour les listes. Comme pour les rendus avec des conditions, tout se gère avec la syntaxe JSX.

En JSX, nous allons pouvoir itérer sur une liste, voire mieux faire une transformation de chaque item de la liste en élément ou composant React.

```
// Liste de données
const numbers = [1, 2, 3, 4, 5];

// Transformation de la liste de données en liste d'éléments React
const listItems = numbers.map((number) =>
  <li>{number}</li>
);

// Evaluation de la liste d'éléments dans le bloc JSX
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

La seule vraie contrainte lorsque l'on manipule des listes est une contrainte technique pour de l'optimisation de performance. Pour rappel, React fonctionne avec un système de Virtual DOM. Lorsqu'il détecte un changement dans un élément du Virtual DOM, il met à jour le DOM. Pour identifier de manière plus optimisée les différents item d'une liste, React préconise l'ajout de l'attribut key sur les composants frères de la liste.

- Reprenez l'application avec la liste de formation.
- Créez une liste de formation comme par exemple :

```
const trainings = [{
  id: 1,
  name : 'React.js'
}];
```

- Ajoutez autant d'items que nécessaire
- Modifiez votre code pour créer la liste à partir de ce tableau
- Tout fonctionne et pourtant vous devez voir apparître ce warning dans la console JavaScript



Warning: Each child **in** an array or iterator should have a unique **"key"** prop.

Check the render method **of** **'TrainingList'**. See <https://fb.me/react-warning-keys> for more information.

**in** TrainingItem (at App.js:29)

**in** TrainingList (at App.js:93)

**in** div (at App.js:88)

**in** App (at index.js:7)

- Ajoutez l'attribut `key` sur le composant `TrainingItem` en utilisant `l'id`. Le warning devrait disparaître

## 4.2. Forms

La gestion des formulaire est un autre élément essentielle d'une application web. Les éléments de formulaire standards de l'API DOM conservent un state interne avec la valeur de l'élément. Il sera nécessaire de 'synchroniser' l'état de l'élément HTML avec l'état du composant React.

Le second impact concerne la validation du formulaire. Le comportement par défaut va naviguer vers une autre page. Une application React étant une application de type SPA, nous souhaitons rester sur la page actuellement mais pouvoir capter l'évènement de validation.

L'exemple ci-dessous illustre les deux points précédent :

```
import React, { Component } from 'react';
import './App.css';

class App extends Component {

  constructor(props) {
    super(props);

    this.state = {email: ''};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    console.log('email changed', event.target.value, event);
    this.setState({email: event.target.value});
  }

  handleSubmit(event) {
    alert('A email was submitted: ' + this.state.email);
    event.preventDefault();
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <div>
          <form onSubmit={this.handleSubmit}>
            <label>
              Email:
              <input type="email" value={this.state.email} onChange={this.
handleChange} />
            </label>
            <input type="submit" value="Submit" />
          </form>
        </div>
      </div>
    );
  }
}

export default App;
```

Voici un autre exemple avec un composant de type select

```
class FlavorForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: 'coconut'};

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({value: event.target.value});
  }

  handleSubmit(event) {
    alert('Your favorite flavor is: ' + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Pick your favorite La Croix flavor:
          <select value={this.state.value} onChange={this.handleChange}>
            <option value="grapefruit">Grapefruit</option>
            <option value="lime">Lime</option>
            <option value="coconut">Coconut</option>
            <option value="mango">Mango</option>
          </select>
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```



Il existe des bibliothèques tierces pour la gestion de formulaires. Une des plus connues est [react-form](#)



Si vous souhaitez faire des validations et afficher des messages d'erreur, vous pouvez détecter les changements de la valeur d'un champ et via une propriété du state gérer l'affichage.

### 4.2.1. TP

- Créez une nouvelle application
- Ajoutez 3 champs de types texte
  - email ⇒ Doit être un email valide sinon affichage d'un message d'erreur
  - password ⇒ Doit comporter 4 caractères
  - confirm password ⇒ Doit être identique au password
- Le bouton de validation sera enable si tous les champs sont valides
- Au click sur le bouton de validation, loguez le state du composant

## 4.3. Composition

React préconise la composition à l'héritage pour mutualiser et rendre génériques de composants.

La propriété `children` des props (props.children) permet de récupérer les éléments ou composants React enfants du composant courant.

```
// Ce composant crée un div, lui applique un style et lui ajoute en composants
enfants, ces propres composants enfants
```

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' + props.color}>
      {props.children}
    </div>
  );
}
```

```
// utilisation du composant FancyBorder dans un autre composant. Les éléments h1
et p seront "copiés"
// dans le composant FancyBorder via props.children
```

```
function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}
```

Cette approche permet de rendre des composants génériques. Cela permet aussi de réaliser des composants "techniques" comme par exemple des conteneurs. Avec des framework orientés composants, nous préférons souvent avoir de petits composants à la responsabilité bien définie. C'est plus simple à écrire, à maintenir et à tester.

Un exemple de conteneur technique que j'utilise dans toutes mes applications est le `ConnectionContainer`. Sa responsabilité est de gérer les informations de connectivité pour l'application. Il va écouter les changements de connectivité, maintenir un state avec la valeur (pour la transmettre à ceux qui en ont besoin) et afficher un label lorsqu'il n'y a plus de connectivité.

```
export class ConnectionContainer extends Component {

  constructor(props) {
    super(props);

    this.state = {
      onLine : navigator.onLine,
    }

    this.addListener();
  }

  addListener () {
    window.addEventListener("offline", () => this.setState({onLine : false}),
false);
    window.addEventListener("online", () => this.setState({onLine : true}),
false);
  }

  render() {

    const offlineContainer = (
      <div>
        <h1>OFFLINE</h1>
      </div>
    );

    return (
      <div>
        <div>
          {this.props.children}
        </div>
        {!this.state.onLine && offlineContainer}
      </div>
    );
  }
}
```

- Intégrez ce composant dans l'application des listes de formation où vous affichez un label ONLINE/OFFLINE.
- Supprimez le bouton, et déclarez le composant ConnectionContainer

## 4.4. PropTypes

Les PropTypes vont nous permettre de définir quels sont les props attendues, requises pour un composant, mais aussi leurs types. Même si le typage dynamique est une des forces du langage JavaScript, il est pratique de valider les props en entrées d'un composant. React lèvera des erreurs si la validation échoue.

Aujourd'hui les proptypes sont dans une librairie npm tierce. Il faut d'abord l'installer avant de pouvoir en bénéficier.

- Installez la [dépendance](#) dans le projet de la liste de formation

```
yarn add prop-types
```

- Ajoutez les proptypes sur les composants que vous avez créés



Vous pouvez regarder le [README](#) du repository Github afin de découvrir les différents validateurs

Il est également possible de définir des valeurs par défaut au props



```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

// Specifies the default values for props:
Greeting.defaultProps = {
  name: 'Stranger'
};

// OU

class Greeting extends React.Component {
  static defaultProps = {
    name: 'stranger'
  }

  render() {
    return (
      <div>Hello, {this.props.name}</div>
    )
  }
}
```



Pour tous les TP's suivants, les validations des props des composants doivent être mises en place, c'est une bonne pratique.

## 4.5. TP : Todo Application

- Créez une nouvelle application et créez un repo Github pour gérer les sources
- Un Todo est défini comme ci-dessous.

```
{  
  "title": "Send a mail",  
  "isDone": false  
}
```

- Générez un composant `TodoList` qui affichera une liste de `TodoItem` et le nombre total de Todos présents dans la liste
- Créez une liste de todos mockés dans un premier temps (4 ou 5)
- Générez un composant `TodoItem` qui affiche le titre en uppercase et une checkbox bindée sur le boolean `isDone` d'un Todo.
- Quand le todo est effectué, l'utilisateur coche la checkbox, ce qui a pour effet de barrer le titre du Todo.
- Générez un composant `TodoForm` qui permet de saisir un Todo et le positionner au dessus du composant `TodoList` dans l'interface. Il sera composé d'un champ de saisie pour le titre et d'un bouton d'ajout. Au click sur le bouton d'ajout, le todo est ajouté à la liste.
- Supprimez la liste mockée maintenant que l'on peut ajouter des todos.
- Quand la liste est vide un texte doit être affiché pour informer l'utilisateur qu'il n'a aucun todo. "Vous n'avez aucun todo"
- Dans le composant `TodoForm`, ajoutez un bouton reset qui réinitialise la liste des todos
- Poussez vos sources dans votre repo git

# Chapitre 5. API

## 5.1. Context

L'API Context permet de passer des données d'un composant parent à un composant enfant non direct (petits enfants, arrières petits enfants). Avec une approche sans l'API Context, il faudrait le faire transiter de composants en composants.

L'API Context va permettre de définir un producteur de données et des consommateurs utilisables uniquement dans les composants enfants du producteur de données.



L'utilisation de l'API Context est utile pour des données globales, un thème, les données de l'utilisateur authentifié, ...

- Reprenez le projet de liste de formation
- Créez un fichier dédié pour le TrainingContext

```
import React from 'react';

export const trainingList = [{
  id: 1,
  name: 'React.js'
}, {
  id: 2,
  name: 'React Native'
}, {
  id: 3,
  name: 'Angular'
}, {
  id: 4,
  name: 'Typescript'
}];

export const TrainingContext = React.createContext({
  trainingList,
  addTraining : () => {}
});
```

Deux propriétés sont déclarées dans le contexte, la liste de données et une fonction pour ajouter une formation à la liste. La liste de données sera ensuite consommée dans le composant TrainingList pour afficher la liste dans l'interface.

Un bouton est aussi ajouté afin de pouvoir ajouter une formation à la liste le contexte.

- Modifiez le TrainingList comme ceci, en ajoutant le consumer

```
import React, { Component } from 'react';
import { TrainingItem } from './TrainingItem';
import { TrainingContext } from './TrainingContext';

export class TrainingList extends Component {

  addTraining () {

  }

  render () {
    return (
      <TrainingContext.Consumer>
      {
        ({trainings, addTraining}) => {
          return (
            <div>
              Formation :
              {
                trainings.map((item) => <TrainingItem key= {item.id} name={item.
name}></TrainingItem>)
              }
            <div>
              <button onClick={() => addTraining('Formation ajoutée via le
contexte')}> Add training</button>
            </div>
          </div>
        )
      }
    </TrainingContext.Consumer>
  )
}
```

- Ensuite le composants App doit fournir le producer à l'arbre de composants

```
import React, {Component} from 'react';
import './App.css';
import {TrainingList} from './TrainingList';
import {ConnectionContainer} from './ConnectionContainer';
import {TrainingContext, trainingList} from './TrainingContext';

class App extends Component {

  constructor(props) {
    super(props);

    this.addTraining = (name) => {
      this.setState((prevState) => ({
        trainings: [...prevState.trainings, {id: prevState.trainings.length + 1,
name}],
      }));
    }

    this.state = {
      trainings: trainingList,
      addTraining: this.addTraining,
    }
  }

  render() {
    return (
      <div className="App">
        <ConnectionContainer>
          <TrainingContext.Provider value={this.state}>
            <header className="App-header">
              <h1 className="App-title">Welcome to React</h1>
            </header>
            <TrainingList></TrainingList>
          </TrainingContext.Provider>
        </ConnectionContainer>
      </div>
    );
  }
}

export default App;
```

### 5.1.1. TP

- Reprenez le projet de l'application Todo.



Si vous n'avez pas eu le temps de le faire et/ou finir, une version est disponible sur la branche master de ce [repository](#)

- Gérer la liste des todos et les différentes actions via l'API Context

## 5.2. Ref - Forwarding ref

Nous avons besoin de temps en temps d'avoir une référence sur un élément ou un composant. Un cas d'usage pourrait être par exemple de donner le focus sur un champ de saisie, ou bien même récupérer la position du scroll dans une liste , ....

### 5.2.1. Ref

- Reprenez le projet que vous avez réalisé avec les formulaires.
- Créez une référence pour l'input du password

```
this.passwordRef = React.createRef();
```

- Sur le lifecycle componentDidMount, donnez le focus au champ password

```
import React, { Component } from 'react';
import './App.css';

const EMAIL_REGEX = /^\\w+@[a-zA-Z_]+?\\. [a-zA-Z]{2,3}$/;
class App extends Component {

  constructor(props) {
    super(props);

    this.state = {
      email: '',
      password: '',
      confirm: '',
    };

    // Création de la référence
    this.passwordRef = React.createRef();

    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  isEmailValid () {
    return this.state.email.match(EMAIL_REGEX);
  }

  isPassordValid () {
    return this.state.password.length >= 4;
  }

  isConfirmValid () {
```

```
    return this.isPassordValid() && this.state.password === this.state.confirm;
  }

  isSubmitEnabled () {
    return this.isConfirmValid() && this.isEmailValid() && this.isPassordValid();
  }

  handleChange(event) {
    const target = event.target;
    const name = target.name;

    console.log(`${name} changed`, target.value);
    this.setState({[name]: target.value});
  }

  handleSubmit(event) {
    alert('A email was submitted: ' + this.state.email);
    event.preventDefault();
  }

  componentDidMount () {
    // Donne le focus à l'input du password
    this.passwordRef.current.focus();
  }

  render() {
    return (
      <div className="App">
        <header className="App-header">
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <div>
          <form onSubmit={this.handleSubmit}>
            <label>
              Email:
              <input
                name="email" type="email"
                value={this.state.email}
                onChange={this.handleChange}
              />
              {!this.isEmailValid() && <div>L'email n'est pas valide</div>}
            </label>
            <label>
              Password:
              <input
                ref={this.passwordRef}
                name="password"
                type="text"
                value={this.state.password}
              />
            </label>
          </form>
        </div>
      </div>
    );
  }
}
```



```

        onChange={this.handleChange}
      />
    </label>
    {!this.isPassordValid() && <div>Le password n'est pas valide</div>}
    <label>
      Confirm:
      <input
        name="confirm"
        type="text"
        value={this.state.confirm}
        onChange={this.handleChange}
      />
    </label>
    {!this.isConfirmValid() && <div>La confirmation du password n'est pas
valide</div>}
    <input type="submit" value="Submit" disabled={!this.isSubmitEnabled()} />
  </form>
</div>
</div>
);
}
}

export default App;

```

### 5.2.2. Forwarding Ref

L'idée de cet API est de pouvoir transmettre une référence d'un élément ou d'un composant à un composant enfant

```

const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} className="FancyButton">
    {props.children}
  </button>
));

```

Il est désormais possible de passer une référence au composant FancyButton

```

// You can now get a ref directly to the DOM button:
const ref = React.createRef();
<FancyButton ref={ref}>Click me!</FancyButton>;

```



C'est surtout très utile dans des techniques avancées telles que le High Order Component. C'est une fonction qui prend en argument un composant et retourne un nouveau composant. Voici le [lien](#) si vous voulez en savoir plus sur le sujet

# Chapitre 6. React Router

## 6.1. Installation

Il n'existe pas de composants ou d'API dans React pour gérer la navigation. IL existe donc des librairies tierces pour le faire. La plus populaire et active est [React Router](#).

- Créez une nouvelle application
- Ajoutez la dépendance React Router

```
yarn add react-router-dom
```

- Dans le fichier src/App.js, remplacez le code par le code suivant

```
import React from 'react'
import {
  BrowserRouter as Router,
  Route,
  Link
} from 'react-router-dom'

const Home = () => (
  <div>
    <h2>Home</h2>
  </div>
)

const About = () => (
  <div>
    <h2>About</h2>
  </div>
)

const Topic = ({ match }) => (
  <div>
    <h3>{match.params.topicId}</h3>
  </div>
)

const Topics = ({ match }) => (
  <div>
    <h2>Topics</h2>
    <ul>
      <li>
        <Link to={`/${match.url}/rendering`}>
```

```

    Rendering with React
    </Link>
  </li>
  <li>
    <Link to={` ${match.url}/components`} >
      Components
    </Link>
  </li>
  <li>
    <Link to={` ${match.url}/props-v-state`} >
      Props v. State
    </Link>
  </li>
</ul>

<Route path={` ${match.path}/:topicId`} component={Topic}/>
<Route exact path={match.path} render={() => (
  <h3>Please select a topic.</h3>
)}>
</div>
)

const BasicExample = () => (
  <Router>
    <div>
      <ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/about">About</Link></li>
        <li><Link to="/topics">Topics</Link></li>
      </ul>

      <hr/>

      <Route exact path="/" component={Home}/>
      <Route path="/about" component={About}/>
      <Route path="/topics" component={Topics}/>
    </div>
  </Router>
)
export default BasicExample

```

- Testez l'application et la navigation

## 6.2. Routes

Nous allons reprendre composant par composant. Dans un premier temps nous allons nous concentrer sur le Router et les Routes.

### 6.2.1. Router

Il y a plusieurs types de routeurs, `BrowserRouter` et `HashRouter`. Tous les deux s'occupent de manager l'historique via un objet `history` créé par React Router.



Generally speaking, you should use a `<BrowserRouter>` if you have a server that responds to requests and a `<HashRouter>` if you are using a static file server.

`BrowserRouter` est le routeur le plus communément utilisé car il s'appuie sur l'API HTML5 history. IL maintient la cohérence entre le champ URL de votre navigateur et l'état de votre application.



Pour rappel, une application React est une SPA. React Router gère la navigation au sein de l'application React mais techniquement, nous n'avons pas changé de page web (HTML). Pour garder une expérience utilisateur qui respecte les standards du web, un changement de route doit se répercuter dans l'URL du navigateur. De même, si un utilisateur veut mettre une page en favori ou accéder à une page en particulier de l'application, il doit pouvoir via l'URL.



En javascript le contexte est gardé tant que l'on ne change pas de page web (HTML). Vous devez être attentif à ce que l'affichage d'une route ne dépende pas de la navigation de l'utilisateur. Une route étant accessible par une URL, l'application peut s'initialiser sur cette route directement.

- Dans le fichier `App.js`, ajoutez le `BrowserRouter`

```
import React from 'react';
import {BrowserRouter} from 'react-router-dom';
import {Home} from './routes/Home';

export default class App extends React.Component {
  render () {
    return (
      <BrowserRouter>
      </BrowserRouter>
    )
  }
}
```

## 6.2.2. Routes

- Déclarez une route qui affiche un composant Home lorsque l'URL match

```
import React from 'react';
import {BrowserRouter, Route} from 'react-router-dom';
import {Home} from './routes/Home';

export default class App extends React.Component {
  render () {
    return (
      <BrowserRouter>
        <div>
          <Route exact path="/" component={Home}/>
        </div>
      </BrowserRouter>
    )
  }
}
```

- Créez le composant Home

```
import React from 'react';

export const Home = () => (
  <div>
    <h1>Home route</h1>
  </div>
);
```

- Dans la console regardez la valeur de props

```
Props :
{match: {...}, location: {...}, history: {...}, staticContext: undefined}
history: {length: 27, action: "POP", location: {...}, createHref: f, push: f, ...}
location: {pathname: "/", search: "", hash: "", state: undefined}
match: {path: "/", url: "/", isExact: true, params: {...}}
```

React Router passe des objets en props d'une route :

- **history** ⇒ objet qui représente l'historique
- **location** ⇒ Objet qui représente l'état courant de l'application
- **match** ⇒ Objet qui représente les données modélisant le matching du path avec l'URL courante



L'objet history est mutable, il est recommandé de récupérer les informations de location via l'objet location et non pas via history.location.

- Faites évoluer la navigation comme ceci :

```
import React from 'react';
import {BrowserRouter, Route} from 'react-router-dom';
import {Home} from './routes/Home';
import {About} from './routes/About';
import {Params} from './routes/Params';
import {Hierarchical} from './routes/Hierarchical';

export default class App extends React.Component {
  render () {
    return (
      <BrowserRouter>
        <div>
          <Route exact path="/" component={Home}/>
          <Route path="/about" component={About}/>
          <Route path="/:custom" component={Params}/>
          <Route path="/hierarchical" component={Hierarchical}/>
        </div>
      </BrowserRouter>
    )
  }
}
```

- Créez les différents composants requis pour la navigation

Sources: 3. router/src/routes-multiples/routes/Home.js

```
import React from 'react';

export const Home = (props) => {

  console.log('Props : ', props);

  return (
    <div>
      <h1>Home page</h1>
    </div>
  )
}
```

Sources: 4. router/src/routes-multiples/routes/About.js

```
import React from 'react';

export const About = () => (
  <div>
    <h1>About route</h1>
  </div>
)
```

Sources: 5. router/src/routes-multiples/routes/Params.js

```
import React from 'react';

export const Params = (props) => {
  return (
    <div>
      <h1>Params route</h1>
      <p>Paramètre : {props.match.params.custom}</p>
    </div>
  )
}
```

Sources: 6. router/src/routes-multiples/routes/Hierarchical.js

```
import React from 'react';
import {Route} from 'react-router-dom';

export function SubRoute ({match}) {
  return (
    <div>
      <h1>`${match.params.name} sub route`</h1>
    </div>
  )
}

export const Hierarchical = ({match}) => {

  return (
    <div>
      <h1>Hierarchical route</h1>
      <div>
        <Route path={`/${match.path}/${match.params.name}`} component={SubRoute}/>
        <Route path={match.path} render={() => (
          <h3>Always display</h3>
        )}/>
      </div>
    </div>
  )
}
```

- Naviguer via les URL du navigateur
  - <http://localhost:3000/about>
  - <http://localhost:3000/niko>
  - <http://localhost:3000/hierarchical>
  - <http://localhost:3000/hierarchical/niko>



Vous devriez constaté que certaines URL match avec plusieurs routes. En effet, `/about` et `/:custom` match le même pattern. Vous pouvez ajouter l'attribut `exact` pour contrer le comportement.



## 6.3. Switch

Précédemment, vous avez été confronté au comportement par défaut des routes. Les routes sont indépendantes, les unes des autres, et sont rendues uniquement en fonction de l'URL et du fait qu'elle match ou pas avec la path définit pour chaque route.

Le composant Switch va nous permettre de rendre une route qui match avec l'URL. Jusque là rien de nouveau !! Mais il va rendre le premier composant qui match avec l'URL.

- Modifiez le fichier App.js en ajoutant le composant Switch

```
import React from 'react';
import {BrowserRouter, Route, Switch} from 'react-router-dom';
import {Home} from './routes/Home';
import {About} from './routes/About';
import {Params} from './routes/Params';
import {Hierarchical} from './routes/Hierarchical';

export default class App extends React.Component {
  render () {
    return (
      <BrowserRouter>
        <div>
          <Switch>
            <Route exact path="/" component={Home}/>
            <Route path="/about" component={About}/>
            <Route path="/:custom" component={Params}/>
            <Route path="/hierarchical" component={Hierarchical}/>
          </Switch>
        </div>
      </BrowserRouter>
    )
  }
}
```

- Testez les URL comme précédemment.
- Que constatez vous ?

## 6.4. Link

React Router met à disposition deux composants pour créer des liens vers des routes de l'application. Sous le capot de ces composants, une balise HTML `<a>` est générée.

### 6.4.1. Link

Voici un exemple de lien

```
import { Link } from 'react-router-dom'  
// Au click sur le clien, le router navigue vers la location /about  
<Link to="/about">About</Link>
```

Il est également possible de passer un objet plutôt qu'un chaîne de caractère

```
<Link to={{  
  pathname: '/courses',  
  search: '?sort=name',  
  hash: '#the-hash',  
  state: { fromDashboard: true }  
}}/>
```

### 6.4.2. NavLink

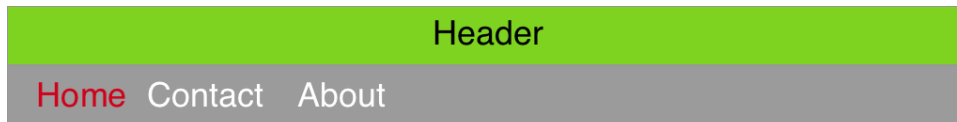
C'est un composant Link spécialisé pour le menu d'une application. La spécificité est assez simple, permettre d'appliquer un style au lien quand la valeur de l'attribut `to` match avec l'URL

```
<NavLink  
  to="/faq"  
  activeClassName="selected"  
>FAQs</NavLink>
```

```
<NavLink  
  to="/faq"  
  activeStyle={{  
    fontWeight: 'bold',  
    color: 'red'  
  }}  
>FAQs</NavLink>
```

## 6.5. TP

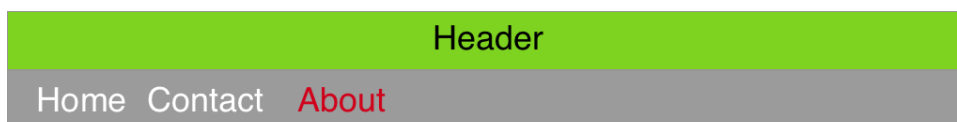
- Créez une application React
- Mettez en place une navigation comme dans les wireframes suivants :
- Home Page



Home Page



- About Page



About Page



- Contact Page Créez une liste de contacts mockées. Quand un contact est sélectionné, affichez

dans une sous route, une fiche de détail

## Header

[Home](#) [Contact](#) [About](#)

Charlie Hardy

Leona Frank

Derek Alexander

Benjamin Parks

Alex Santos

Charlie Hardy

7777 Kilback Cove  
Hilpertfort  
Zimbabwe  
karley\_ernser@yahoo.com

## Footer

# Chapitre 7. Redux

## 7.1. Actions synchrones

- Reprenez le projet sur avec les listes de formations. Nous allons le passer avec une architecture Redux
- Installez les dépendances nécessaires

```
yarn add react-redux redux redux-logger prop-types
```

- Créez les actions

Sources: 7. src/redux/store/trainings.action.js

```
export const ADD_TRAINING = 'ADD_TRAINING';

export function addTraining(name) {
  return {
    type: ADD_TRAINING,
    name
  }
}
```

- Créez le reducer

Sources: 8. src/redux/store/trainings.reducer.js

```
import {ADD_TRAINING} from './trainings.action';

export const trainingList = [{
  id: 1,
  name: 'React.js'
}, {
  id: 2,
  name: 'React Native'
}, {
  id: 3,
  name: 'Angular'
}, {
  id: 4,
  name: 'Typescript'
}];

export const initialState = {
  list: trainingList,
};

export function trainingsReducer (state = initialState, action) {
  switch (action.type) {
    case ADD_TRAINING:

      const training = {
        id: state.list.length + 1,
        name: action.name,
      }

      return {
        ...state,
        list: [...state.list, training],
      }
    default :
      return state;
  }
}
```

- Créez le store

Sources: 9. src/redux/App.js

```
import React, {Component} from 'react';
import { combineReducers, createStore, applyMiddleware } from 'redux';
import { Provider } from 'react-redux';
import {createLogger} from 'redux-logger';
import './App.css';
import TrainingList from './TrainingList';
import {ConnectionContainer} from './ConnectionContainer';
import {trainingsReducer} from './store/trainings.reducer';

// Assemblage des différents reducers d'une application
const reducers = combineReducers({
  trainings: trainingsReducer,
});

const logger = createLogger({
  level: 'log',
});

// Création du store
const store = createStore(reducers, applyMiddleware(logger));

class App extends Component {

  render() {
    return (
      <Provider store={store}>
        <div className="App">
          <ConnectionContainer>
            <header className="App-header">
              <h1 className="App-title">Welcome to React</h1>
            </header>
            <TrainingList></TrainingList>
          </ConnectionContainer>
        </div>
      </Provider>
    );
  }
}

export default App;
```

- Branchez le composant TrainingList au store pour récupérer la liste des formation mais aussi l'action pour ajouter une formation

Sources: 10. src/redux/TrainingList.js

```
import React, { Component } from 'react';
import { connect } from 'react-redux';
import { TrainingItem } from './TrainingItem';
import PropTypes from 'prop-types';
import { addTraining } from './store/trainings.action';

export class TrainingList extends Component {

  render () {
    return (
      <div>
        Formation :
        {
          this.props.trainings.map((item) => <TrainingItem key= {item.id} name={item
.name}></TrainingItem>)
        }
        <div>
          <button onClick={() => this.props.addTraining('formation ajoutée via
redux')}> Add training</button>
        </div>
      </div>
    )
  }
}

TrainingList.propTypes = {
  addTraining: PropTypes.func.isRequired,
  trainings: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      name: PropTypes.string.isRequired,
    })
  ).isRequired,
}

const mapStateToProps = state => ({
  trainings: state.trainings.list,
});

const mapDispatchToProps = dispatch => ({
  addTraining: name => dispatch(addTraining(name))
})

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(TrainingList)
```



## 7.2. Actions asynchrones avec Redux-thunk

Dans l'implémentation précédente, toutes les actions étaient synchrones. Dans la réalité, un certain nombre de traitements sont asynchrones comme par exemple les appels réseaux. Les traitements asynchrones sont une bonne pratique en JavaScript car le langage l'encourage, mais aussi, pour des questions de performances. En effet, le UI thread pourrait être bloqué par un process synchrone ce qui provoquerait une expérience utilisateur dégradée, une interface qui se fige ou qui lague dans les animations, ..

**Redux Thunk** est une librairie tierce qui répond au besoin. Ce n'est pas la seule et il y a des pros et cons, mais c'est celle que l'on manipule souvent en premier.



Il existe aussi d'autres approches plus orientée [programmation réactive](#). Pour aller plus loin, vous pourrez regarder à vous heures perdues des outils tels que [[https://redux-saga.js.org/Redux Saga](https://redux-saga.js.org/Redux%20Saga)] ou [Redux Observable](#).

### Qu'est-ce qu'un thunk ?



Un thunk est une fonction qui enveloppe une expression pour retarder son évaluation.

```
// calculation of 1 + 2 is immediate
// x === 3
let x = 1 + 2;

// calculation of 1 + 2 is delayed
// foo can be called later to perform the calculation
// foo is a thunk!
let foo = () => 1 + 2;
```



React Thunk est un [middleware Redux](#)

- Installez `redux-thunk`

```
yarn add redux-thunk
```

- Déclarez le middleware thunk

Sources: 11. src/redux-thunk/App.js

```
import React, {Component} from 'react';
import { combineReducers, createStore, applyMiddleware } from 'redux';
import { Provider } from 'react-redux';
import './App.css';
import TrainingList from './TrainingList';
import {ConnectionContainer} from './ConnectionContainer';
import {trainingsReducer} from './store/trainings.reducer';
import {createLogger} from 'redux-logger';
import thunk from 'redux-thunk';

// Assemblage des différents reducers d'une application
const reducers = combineReducers({
  trainings: trainingsReducer,
});

const logger = createLogger({
  level: 'log',
});

// Création du store
const store = createStore(reducers, applyMiddleware(thunk, logger));

class App extends Component {

  render() {
    return (
      <Provider store={store}>
        <div className="App">
          <ConnectionContainer>
            <header className="App-header">
              <h1 className="App-title">Welcome to React</h1>
            </header>
            <TrainingList></TrainingList>
          </ConnectionContainer>
        </div>
      </Provider>
    );
  }
}

export default App;
```

- Créez une fonction qui retourne une promise avec un setTimeout pour simuler un process asynchrone.

Sources: 12. src/redux-thunk/store/trainings.action.js

```
export const ADD_TRAINING = 'ADD_TRAINING';
export const TRAININGS_LOADED = 'TRAININGS_LOADED';
export const LOAD_TRAININGS = 'LOAD_TRAININGS';

const data = [{
  id: 1,
  name: 'React.js'
}, {
  id: 2,
  name: 'React Native'
}, {
  id: 3,
  name: 'Angular'
}, {
  id: 4,
  name: 'Typescript'
}]

const mockFetch = () => {
  return new Promise((resolve) => {
    setTimeout(() => resolve(data), 2000)
  })
}

export function addTraining(name) {
  return {
    type: ADD_TRAINING,
    name
  }
}

export function trainingLoaded (datas) {
  return {
    type : TRAININGS_LOADED,
    trainings : datas,
  }
}

export function loadTrainings () {
  return (dispatch) => {

    // Dispatch load trainings start
    dispatch({type: LOAD_TRAININGS});

    return mockFetch()
      .then((datas) => dispatch(trainingLoaded(datas)));
  }
}
```

- Mettez le reduceur à jour avec les actions précédemment créées

Sources: 13. `src/redux-thunk/store/trainings.reducer.js`

```
import {ADD_TRAINING, LOAD_TRAININGS, TRAININGS_LOADED} from './trainings.action';

export const initialState = {
  list: [],
  loading: false,
  loaded: false,
};

export function trainingsReducer(state = initialState, action) {
  switch (action.type) {
    case LOAD_TRAININGS:
      return {
        ...state,
        loading: true,
        loaded: false,
      }
    case TRAININGS_LOADED:
      return {
        ...state,
        list: action.trainings,
        loading: false,
        loaded: true,
      }
    case ADD_TRAINING:
      const training = {
        id: state.list.length + 1,
        name: action.name,
      }
      return {
        ...state,
        list: [...state.list, training],
      }
    default:
      return state;
  }
}
```

- Et enfin le composant TrainingList

Sources: 14. `src/redux-thunk/TrainingList.js`

```
import React, { Component } from 'react';
import {connect} from 'react-redux';
import {TrainingItem} from './TrainingItem';
```

```

import PropTypes from 'prop-types'
import {addTraining, loadTrainings} from './store/trainings.action';

export class TrainingList extends Component {

  componentDidMount() {
    this.props.loadTrainings();
  }

  displayTrainings () {
    if (this.props.loading) {
      return <p>Loading ....</p>
    }

    return this.props.trainings.map((item) => <TrainingItem key= {item.id} name={item
.name}></TrainingItem>);
  }

  render () {
    return (
      <div>
        Formation :
        {
          this.displayTrainings()
        }
        <div>
          <button onClick={() => this.props.addTraining('formation ajoutée via redux'
)}> Add training</button>
        </div>
      </div>
    )
  }
}

TrainingList.propTypes = {
  addTraining: PropTypes.func.isRequired,
  loadTrainings: PropTypes.func.isRequired,
  loading: PropTypes.bool.isRequired,
  trainings: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      name: PropTypes.string.isRequired,
    })
  ).isRequired,
}

const mapStateToProps = state => ({
  trainings: state.trainings.list,
  loading: state.trainings.loading,

```

```
});  
  
const mapDispatchToProps = dispatch => ({  
  addTraining: name => dispatch(addTraining(name)),  
  loadTrainings: () => dispatch(loadTrainings()),  
})  
  
export default connect(  
  mapStateToProps,  
  mapDispatchToProps  
) (TrainingList)
```

## 7.3. TP

- Reprenez le projet de l'application Todo.



Si vous n'avez pas eu le temps de le faire et/ou finir, une version est disponible sur la branche master de ce [repository](#)

- Gérer la liste des todos et les différentes actions avec Redux.

# Chapitre 8. Code quality

## 8.1. Eslint

Eslint est ce que l'on appelle un **code style checker**. Il permet de valider que notre code respecte des règles. Ces règles peuvent aussi bien être, une vérification de la présence du point virgule en fin de ligne comme le respect de règle de nommage, ....

Souvent la première réflexion quand on présente ce genre d'outil à des développeurs qui ont de l'expérience sur d'autres type de langage que le JavaScript, ils sourient ;) En effet, ils pensent souvent que c'est dû au langage car il est interprété et possède un typage dynamique. Alors c'est en parti vrai et en parti faux :D

Effectivement, le JavaScript ne possédant pas de phase de compilation, il n'y a pas de vérification et il est possible d'avoir des soucis au runtime. Mais d'un autre côté, Eslint permet de définir des règles comme celles que l'on pourrait trouver sur un Sonar, il comble certains de ces manques sans en avoir la puissance. Vous trouvez aussi des style checker sur des langages typés tel que Java, Android, Swift, ....

Eslint est donc un outil qui permet de définir des règles de validation. Il est possible de faire des plugins ou des ensembles de règles qui peuvent être externes. Vous trouverez comment créer des règles sur le site de [Eslint](#)

### 8.1.1. react-scripts

React-script est un module utilitaire pour React. Si vous regardez un package.json d'une application React, vous le verrez en dépendance mais aussi dans les balises scripts pour déclarer des lanceurs



```
{
  "name": "code-quality",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "prop-types": "^15.6.1",
    "react": "^16.3.2",
    "react-dom": "^16.3.2",
    "react-redux": "^5.0.7",
    "react-scripts": "1.1.4",
    "redux": "^4.0.0",
    "redux-logger": "^3.0.6",
    "redux-thunk": "^2.2.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "lint": "eslint --ext=js --ext=jsx . --fix",
    "test": "react-scripts test --env=jsdom",
    "eject": "react-scripts eject"
  },
  "devDependencies": {
    "eslint-config-airbnb": "^16.1.0",
    "react-test-renderer": "^16.3.2"
  }
}
```

React-script gère une grande partie des besoins d'une application React, un serveur de développement, le build, les tests. Sous le capot, vous trouverez des outils dont vous avez peut être déjà entendu parler, webpack, babel, jest, et eslint.

D'ailleurs vous avez sûrement déjà dû le rencontrer, dans les logs avec le serveur de développements. Lorsqu'une règle n'est pas respectée, un message est publié.



La plupart des IDE possèdent une intégration (interne ou par plugins) de Eslint pour permettre de visualiser les erreurs dans l'IDE. Tous ces outils se basent sur un fichier de configuration (.eslintrc) qui doit être présent à la racine du projet. Pourtant lorsque l'on crée un projet avec `create-react-app`, ce fichier n'est pas créé. Vous devez le faire vous même.

```
{
  "extends": ["react-app", "airbnb"]
}
```

### 8.1.2. Airbnb rules

Le [JavaScript Style Guide](#) de Airbnb est une référence dans le monde JavaScript. Il y en a d'autres comme par exemple un par Google, mais celui-ci est complet et populaire. Il existe aussi un plugin eslint qui reprends toutes ces règles, [eslint-config-airbnb](#), nous allons l'ajouter à notre projet de formations

- Créez un fichier `.eslintrc` comme précédemment

-Installez les dépendances node manquantes

```
yarn add eslint-config-airbnb -D
```



Eslint et d'autres plugins sont déjà installés par dépendances transitives avec la dépendance react-scripts. C'est pourquoi il y a que le plugin airbnb à installer

- Dans le `package.json`, ajoutez le lanceur

```
"lint": "eslint --ext=js --ext=jsx . --fix",
```



On check les fichiers avec l'extension `js` et `jsx` de tout le répertoire courant récursivement. L'option `--fix` permet de corriger les erreurs que Eslint sait réparer. Ca ne corrige pas tout, loin de la, mais ça aide ;)

- Exécutez eslint

```
npm run lint
```

- Amusez vous à corriger les erreurs !!!

## 8.2. Jest

Jest un outil pour exécuter des tests unitaires. Il a été open source par Facebook et est donc l'utilisateur de référence pour une application React. En tout cas c'est ce qu'installe `create-react-app`.

D'ailleurs un test est présent par défaut dans le squelette d'une nouvelle application.

```
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);
  ReactDOM.unmountComponentAtNode(div);
});
```

Ce test simple vérifié que l'application App peut être montée dans le DOM.

- Jouez le test

```
npm run test
```

- Créez un fichier dans le même dossier que le fichier `trainings.reducer.js` et nommez le `trainings.reducer.test.js`

```
import { trainingsReducer, initialState } from './trainings.reducer';
import { LOAD_TRAININGS } from './trainings.action';

describe('trainings.reducer.js', () => {
  it('should validate state after default action.type', () => {
    expect(trainingsReducer(initialState, { type: 'NOT_VALID' })).toEqual(
      initialState);
  });

  it('should validate state after LOAD_TRAININGS action', () => {
    const state = {
      ...initialState,
      loading: true,
      loaded: false,
    };
    expect(trainingsReducer(initialState, { type: LOAD_TRAININGS })).toEqual(state);
  });
});
```

- Exécutez les tests
- Ajoutez les tests pour les types d'actions TRAININGS\_LOADED et ADD\_TRAINING



Documentation pour l'API [expect](#)

### 8.2.1. Snapshot testing

Le Snapshot testing permet de valider que l'UI d'un composant ne change pas sans qu'on le veuille. Cela va nous permettre de rendre un composant dans un fichier de type .snap. Lorsque vous ferez un nouveau développement qui modifiera votre composant, le fichier .snap ne contiendra plus la même représentation car cela vient de changer. Le développeur devra alors valider par une action si l'évolution est souhaitée.



Les fichiers .snap générés sont à joindre avec les sources dans le gestionnaire de sources

- Installez la dépendance `react-test-renderer`

```
yarn add react-test-renderer -D
```

- Créez un fichier de test pour le composant TrainingItem

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

describe('TrainingItem.js', () => {
  it('should take a snapshot', () => {
    const tree = renderer
      .create(<TrainingItem name="Nicolas" />)
      .toJSON();
    expect(tree).toMatchSnapshot();
  });
});
```

- Modifier le render du composant TrainingItem et relancez le test

```
render() {  
  return (  
    <div>  
      <p onClick={this.onItemClicked}>{this.props.name}</p>  
    </div>  
  );  
}
```

1 snapshot test failed in 1 test suite. Inspect your code changes or press 'u' to update them.

- Si vous souhaitez valider le nouveau snapshot, faut appuyer sur la touche u ;)



Le snapshot testing est une solution portée par Jest. Si on se réfère à la [documentation React](#), Il préconise l'utilisation du projet Enzyme pour le test de composant.

## 8.3. Enzyme

**Enzyme** est module Node.js développé par Airbnb et open source. Enzyme est un utilitaire de test JavaScript pour React qui facilite les tests des composants React.



Enzyme's API is meant to be intuitive and flexible by mimicking jQuery's API for DOM manipulation and traversal.

La vraie plus value de Enzyme est que l'on peut tester un composant de manière unitaire, sans que ses potentiels enfants ne soient pris en compte ⇒ shallow rendering. Enzyme possède 3 possibilités pour rendre les composants :

- **shallow rendering** ⇒ Idéal pour tester des composants sans prendre en compte ses enfants

```
import { shallow } from 'enzyme';
import sinon from 'sinon';

describe('<MyComponent />', () => {
  it('should render three <Foo /> components', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.find(Foo)).toHaveLength(3);
  });

  it('should render an `icon-star`', () => {
    const wrapper = shallow(<MyComponent />);
    expect(wrapper.find('.icon-star')).toHaveLength(1);
  });

  it('should render children when passed in', () => {
    const wrapper = shallow((
      <MyComponent>
        <div className="unique" />
      </MyComponent>
    ));
    expect(wrapper.contains(<div className="unique" />)).toEqual(true);
  });

  it('simulates click events', () => {
    const onClick = sinon.spy();
    const wrapper = shallow(<Foo onClick={onClick} />);
    wrapper.find('button').simulate('click');
    expect(onClick.calledOnce).toEqual(true);
  });
});
```

- **full rendering** ⇒ Idéal pour les composants qui doivent interagir avec l'API DOM ou ceux qui ont besoin des lifecycle hooks pour tester le composant

```
import { mount } from 'enzyme';
import sinon from 'sinon';
import Foo from './Foo';

describe('<Foo />', () => {
  it('calls componentDidMount', () => {
    sinon.spy(Foo.prototype, 'componentDidMount');
    const wrapper = mount(<Foo />);
    expect(Foo.prototype.componentDidMount.calledOnce).to.equal(true);
  });

  it('allows us to set props', () => {
    const wrapper = mount(<Foo bar="baz" />);
    expect(wrapper.props().bar).to.equal('baz');
    wrapper.setProps({ bar: 'foo' });
    expect(wrapper.props().bar).to.equal('foo');
  });

  it('simulates click events', () => {
    const onClick = sinon.spy();
    const wrapper = mount(
      <Foo onClick={onClick} />
    );
    wrapper.find('button').simulate('click');
    expect(onClick.calledOnce).to.equal(true);
  });
});
```

- **static rendering** ⇒ Rends le composants en HTML et analyse le résultat.

```
import React from 'react';
import { render } from 'enzyme';
import PropTypes from 'prop-types';

describe('<Foo />', () => {
  it('renders three `.foo-bar`s', () => {
    const wrapper = render(<Foo />);
    expect(wrapper.find('.foo-bar')).toHaveLength(3);
  });

  it('rendered the title', () => {
    const wrapper = render(<Foo title="unique" />);
    expect(wrapper.text()).toContain('unique');
  });

  it('can pass in context', () => {
    function SimpleComponent(props, context) {
      const { name } = context;
      return <div>{name}</div>;
    }
    SimpleComponent.contextTypes = {
      name: PropTypes.string,
    };

    const context = { name: 'foo' };
    const wrapper = render(<SimpleComponent />, { context });
    expect(wrapper.text()).toEqual('foo');
  });
});
```

- Installez les dépendances

```
yarn add enzyme enzyme-adapter-react-16 react-test-renderer
```

- Modifiez le test du composants TrainingItem



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

describe('TrainingItem.js', () => {
  it('should shallow render', () => {
    shallow(<TrainingItem name="Nicolas" />);
  });

  it('should render the props name', () => {
    const wrapper = shallow(<TrainingItem name="Nicolas" />);
    expect(wrapper.containsMatchingElement(<p>Nicolas</p>)).toEqual(true);
    expect(wrapper.containsMatchingElement(<p>xxxxxx</p>)).toEqual(false);
  });
});
```

- Ajoutez un test au composant TrainingList qui vérifie que l'on a bien 3 TrainingItem si on passe une liste de 3 Trainings props.

## Chapitre 9. Annexes