**React**

[**https://www.taniarascia.com/getting-started-with-react/**](https://www.taniarascia.com/getting-started-with-react/)

[**https://hackernoon.com/react-js-a-better-introduction-to-the-most-powerful-ui-library-ever-created-ecd96e8f4621**](https://hackernoon.com/react-js-a-better-introduction-to-the-most-powerful-ui-library-ever-created-ecd96e8f4621)

[**https://reactjs.org/docs/add-react-to-a-website.html**](https://reactjs.org/docs/add-react-to-a-website.html)

**React projects Github :** [**https://github.com/enaqx/awesome-react#file-uploads**](https://github.com/enaqx/awesome-react#file-uploads)

**Authentication JWT Spring React: https://www.devdiaries.net/blog/Spring-Boot-2-PostgreSQL-JWT-React-Part-3/**

**Definition :**

ReactJS is an open-source, component based front end library responsible only for the **view layer** of the

application. It is maintained by Facebook.

ReactJS uses **virtual DOM** based mechanism to fill in(to put) data (views) in HTML DOM. The virtual DOM works fast owning to the fact that it only changes individual DOM elements instead of reloading complete DOM every time.

**Component states** are what give **React** its **name**. Any time a component’s state changes, its “render” function is called, updating the output. In essence, each component“reacts” to changes, which is handy for any user interface. Data stored in a state should be information that will be updated by the component’s event handlers (changes that should update in real time as the interface is being used).

Stateless: component without state, more reusable

Stateful: component with state

**Squelette index.html:**

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Hello React!</title>

**<script src="https://unpkg.com/react@16/umd/react.development.js"></script>**

**<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>**

**<script src="https://unpkg.com/babel-standalone@6.26.0/babel.js"></script>**

</head>

<body>

**<div id="root"></div>**

**<script type="text/babel">**

**// React code will go here**

**</script>**

</body>

</html>

* [React](https://reactjs.org/docs/react-api.html) – the React top level API
* [React DOM](https://reactjs.org/docs/react-dom.html) – adds DOM-specific methods
* [Babel](https://babeljs.io/) – a JavaScript compiler that lets us use ES6+ in old browsers

You’ll also notice the **text/babel** script type, which is mandatory for using Babel.

**Render:**

the [render()](https://reactjs.org/docs/react-component.html#render) method, the only required method in a class component, which is used to render DOM nodes.

<script type="text/babel">

class **App** extends React.Component {

**render**() {

return (

<h1>Hello world!</h1>

);

}

}

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

</script>

## **JSX: JavaScript + XML that looks like HTML**

const heading = <h1 className="site-heading">Hello, React</h1>;

* className au lieu de class
* onclick will become onClick
* All tags should be closed

**Create Project:**

npm install -g create-react-app

create-react-app reactProj

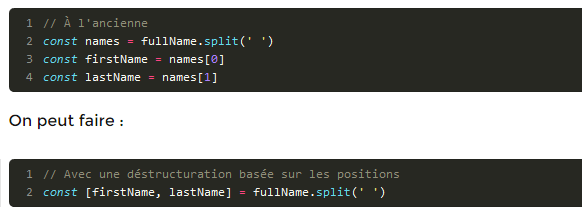
cd reactproj

npm start

* Localhost : 3000



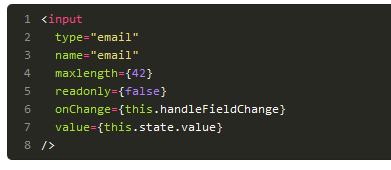
**Split :**





Declaration des props (attribut dans React s’appelle props) :

Seule String s’ecrit de cette facon. Les autres types s’ecrivent entre {}



Remarque : **ClassName** au lieu de Class. Et **HtmlFor** au lieu de For

Les composants On les commence par majuscule sinon c’est considerer juste comme un String :

[

<CoolComponent/>, => V

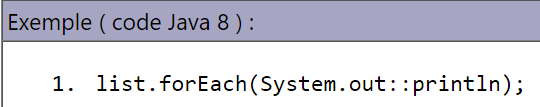
<coolComponent/>, => F

]

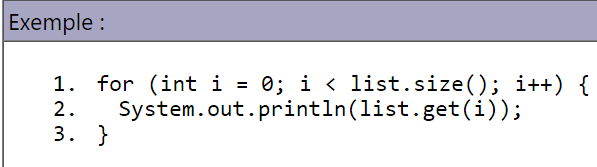
Commentaire :

**{/\*** La classe 'field' assure l’espacement vertical convenable **\*/}**

Expressions Fonctionelles :



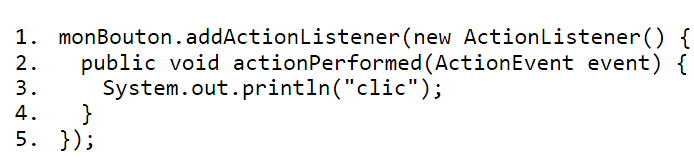
Au lieu de :



Expression Lambda :

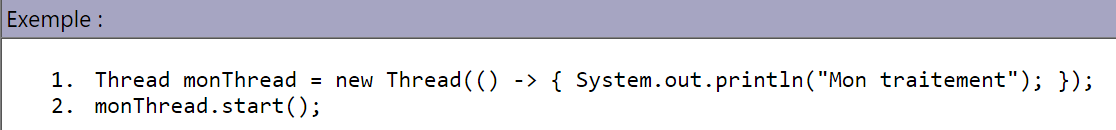


Au lieu de :

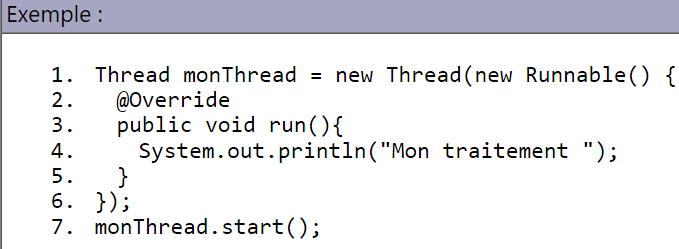


Une expression lambda est utilisée pour représenter une interface fonctionnelle sous la forme d'une expression de la forme :

(arguments) -> corps



Au lieu de :



**Labed :**

Une expression lambda est un raccourci syntaxique aux classes anonymes internes pour une interface qui ne possède qu'une seule méthode abstraite. Ce type d'interface est **nommé interface fonctionnelle**.

**Syntaxe :**

Un des avantages des expressions lambda est d'avoir une syntaxe très simple.

La syntaxe d'une expression lambda est composée de trois parties :

* un ensemble de paramètres, d'aucun à plusieurs
* l'opérateur ->
* le corps de la fonction

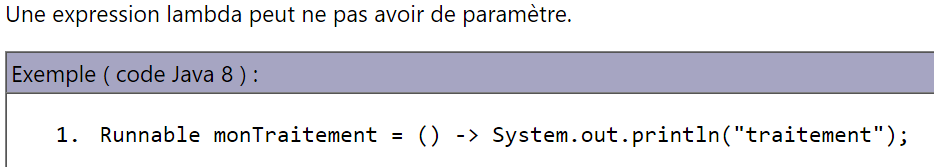
Elle peut prendre deux formes principales :

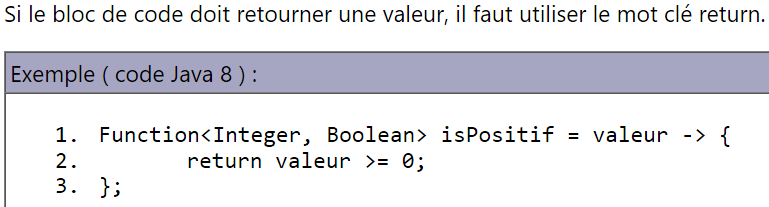
(paramètres) -> expression;

(paramètres) -> { traitements; }

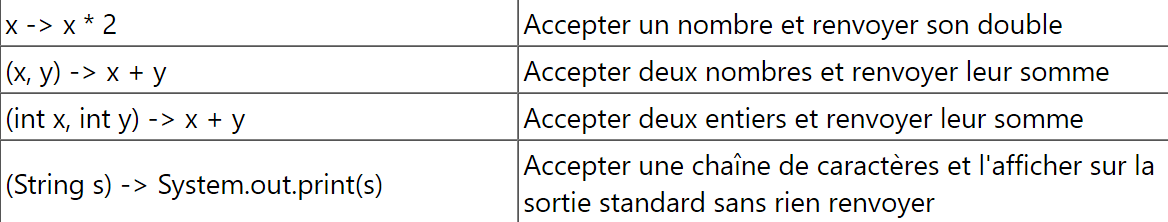
L'écriture d'une expression lambda doit respecter plusieurs règles générales :

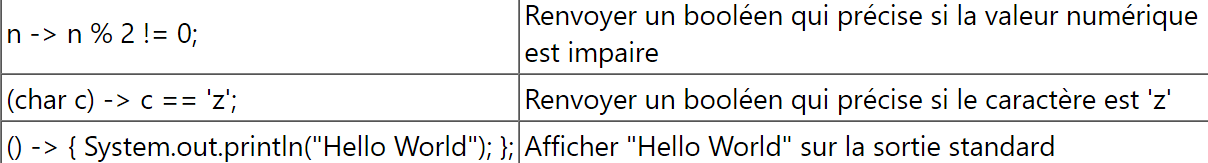
* zéro, un ou plusieurs paramètres dont le type peut être déclaré explicitement ou inféré par le compilateur selon le contexte
* les paramètres sont entourés par des parenthèses et séparés par des virgules. Des parenthèses vides indiquent qu'il n'y a pas de paramètre
* lorsqu'il n'y a qu'un seul paramètre et que son type est inféré alors les parenthèses ne sont pas obligatoires
* le corps de l'expression peut contenir zéro, une ou plusieurs instructions. Si le corps ne contient d'une seule instruction, les accolades ne sont pas obligatoires et le type de retour correspond à celui de l'instruction. Lorsqu'il y a plusieurs instructions alors elles doivent être entourées avec des accolades

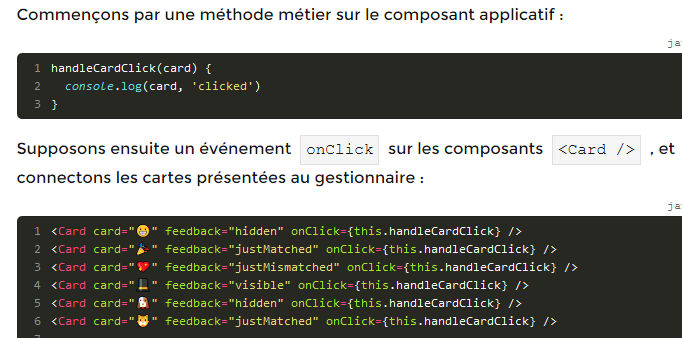




Exemples :









**Remarque** :

Il faut tout d’abord garder en mémoire que JSX produit une **expression** JavaScript et non une instruction. Par conséquent, certaines parties du langage qui constituent des instructions, comme un  if  , un  for  ou une déclaration, n’y sont pas possibles.

L’astuce consiste à remplacer nos instructions classiques de branchement conditionnel par des expressions utilisant les opérateurs logiques (typiquement, &&  ,  ||  et **l’opérateur** **ternaire**  ? : ).

**« Si… Alors… »**

Selon la sémantique de base de JavaScript, si la condition échoue, l'opérande de droite ne sera pas évaluée, et vaudra au global  false . JSX ignorera alors ce  false et rien n’apparaîtra dans le DOM navigateur à cet endroit.

<p>{42 > 43 && document.nonExistingMethod()}</p>

Dans le code ci-dessus, le paragraphe sera vide, l’expression s’arrêtant à son premier opérande, qui vaut  false  , et ce dernier étant automatiquement ignoré par JSX. À l’inverse, si la condition est remplie, l’expression évaluera son opérande de droite, qui constituera alors la valeur finale : notre grappe JSX sera donc utilisée

**« Si… Alors… Sinon… »**

Si nous voulons l’équivalent d’un  if…else…  , nous utiliserons plutôt l’opérateur ternaire  ?…:  , qui est l’équivalent naturel sous forme d’expression. Par exemple :

<p>{user.loggedIn ? <WelcomeLabel /> : <LoginLink />}</p>

<p>{user.admin ? (

<a href="/admin">Faire des trucs de VIP</a>

) : (

<a href="/request-admin">Demander à devenir VIP</a>

)}</p>

La reutilisation de bouton : on le declare comme const et on l’utilise autant de fois :



**Splice :**

The **splice()** method **changes the contents of an array by removing or replacing** existing elements and/or adding new elements in place.

const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb');

// inserts at index 1

console.log(months);

// expected output: Array ["Jan", "Feb", "March", "April", "June"]

-----------------

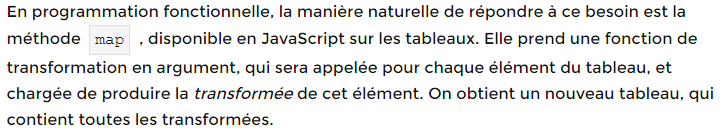
months.splice(4, 1, 'May');

// replaces 1 element at index 4

console.log(months);

// expected output: Array ["Jan", "Feb", "March", "April", "May"]

**Map :**

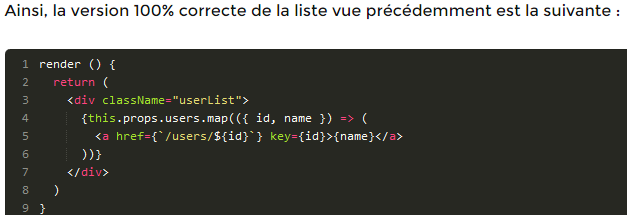


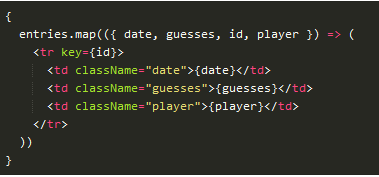






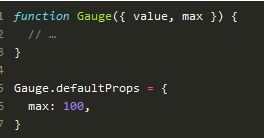
Remarque : React exige que tout élément présent au sein d’un tableau dans une grappe JSX soit doté d’une prop technique nommée  key  . Cette prop doit impérativement être :

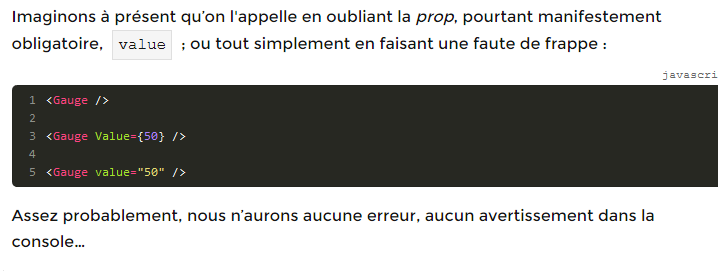




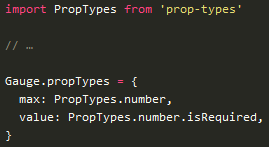
Autre fonctions : Filter, every, some, reduce

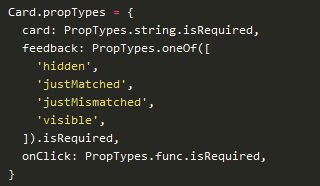
**DefaultProps** : permet de donner des val par defaut aux props





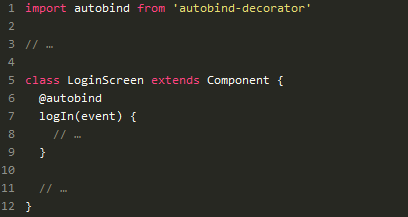
**PropTypes : pour definer le type des props afin de ne pas avoir d’erreur si on se trompe de nommage comme ci dessus**



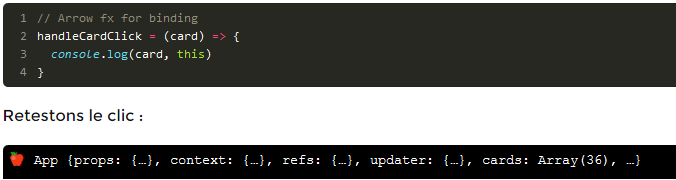


Pour verification cmd:

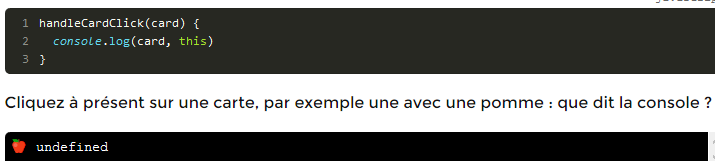
npm install --save prop-types



This :



Au lieu de :



Dans JS : les vaiables simples : passage par valeur => pas de changement de valeur apres passage par fct

Les objets ( function, Array, Literaux) : passage par reference => changement de valeur apres …

This :

State :

Asynchrone : JS est synchrone( un seul thread à la fois) mais on peut realiser l’asynchrone grace au promesses (une promesse d’appeler une fctionnalite au moment le plus opportun) suivantes : les fcts callbacks, yield, async function, ou aussi grace à Ajax

=== :

Un formulaire est controlé :

Si ces champs sont controlés : comportent **value** et **onChange** et pour les cases a cocher **checked** au lieu de value

Un champ n’est pas controlé s’il ne possede pas d’attribut **Value** ou **Checked**

**Pourquoi choisir le Controlé :**

On souhaite filtrer les caractères saisis à la volée

On souhaite formater la valeur à la volée

On choisi le non Controlé quand c’est l’inverse

**import** {Grid, Row, Col} **from** 'react-bootstrap';

<Grid>  
 <Row className="show-grid">  
 <Col xs={12} md={3}>  
 <div className="Logo">  
 </div>  
 </Col>  
 <Col xs={6} md={6}>  
 <ul className="Topnav">  
 <li><a href="#Men" className="Cool-link">Men</a></li>  
 <li><a href="#Womwn" className="Cool-link">Women</a></li>  
 <li><a href="#About" className="Cool-link">About</a></li>  
 </ul>  
 </Col>  
 <Col xs={6} md={3}>  
 <div className="Logo-Panier"></div>  
 <div className="Logo-Login"></div>  
 </Col>  
 </Row>  
</Grid>

Mount a component:

"mounting" (adding nodes to the DOM), "unmounting" (removing them from the DOM), and "updating" (making changes to nodes already in the DOM)

**Exemple**:

**componentWillMount** () {

console.log("componentWillMount")

}

**componentDidMount** () {

console.log("componentDidMount")

}

**render** () {

console.log("render")

...

}

**Result**:

componentWillMount

render

componentDidMount

Redux : https://codeburst.io/redux-a-crud-example-abb834d763c9

Redux is a state management library that gives you access to the state anywhere in your components without the need to pass props. So it can be used with any front-end libraries like Angular and React but it works best with React. ‘react-redux’ is the official library that connects the two.

* npm install --save redux react-redux

First thing to understand about Redux is something called the **store**. It’s where the entire state of your application will live. Instead of having to manage the state in different components we have to only manage it in one single place called the store.c

Bind:

This is not React-specific behavior; it is a part of [how functions work in JavaScript](https://www.smashingmagazine.com/2014/01/understanding-javascript-function-prototype-bind/). Generally, if you refer to a method without () after it, such as onClick={this.handleClick}, you should bind that method.

**class Toggle extends React.Component {**

**constructor(props) {**

**super(props);**

**this.state = {isToggleOn: true};**

**// This binding is necessary to make `this` work in the callback**

**this.handleClick = this.handleClick.bind(this);**

**}**

**handleClick() {**

**this.setState(state => ({**

**isToggleOn: !state.isToggleOn**

**}));**

**}**

**render() {**

**return (**

**<button onClick={this.handleClick}>**

**{this.state.isToggleOn ? 'ON' : 'OFF'}**

**</button>**

**);**

**}**

**}**

The onblur event occurs when an object loses focus.

**Redux:** <https://www.valentinog.com/blog/redux/>

Redux store methods: You might be surprised to know that Redux itself is a small library (2KB) and the **most important methods are just three**:

* **getState** for reading the current state of the application
* **dispatch** for dispatching an action
* **subscribe** for listening to state changes

After learning Redux I realized it wasn't so complex. I knew how to access the current state with getState. I knew how to dispatch an action with dispatch and how to listen for state changes with subscribe.

NB:

Now the initial state is left pristine and **the resulting state is just a copy of the initial state**. Remember two key points for **avoiding mutations in Redux**:

* use concat, slice, or the spread operator for arrays
* use Object.assign or object spread of objects

if (action.type === ADD\_ARTICLE) {

return Object.assign({}, state, {

articles: state.articles.concat(action.payload)

});

}

return state;

OR:

switch (action.type) {

case actions.GET\_POSTS:

return { ...state, loading: true }

case actions.GET\_POSTS\_SUCCESS:

return { posts: action.payload, loading: false, hasErrors: false }

case actions.GET\_POSTS\_FAILURE:

return { ...state, loading: false, hasErrors: true }

default:

return state

}

**Admin:**

Save reference V

List, modify reference V

delete reference V

List, modify, delete sale

**Client:**

List reference V

product detail V

Add to bag, List, modify, delete localstorage V

Add to bag, List, modify, delete DB when authenticated V

Create Account, Authentication V

filter: price range, size, color, marque, SubCategory V

Products | Men | Jacket | Detail product (without click) V

User profil : update profil + log out + sales + redirection + correct sign up (en cours)

Buy operation, sale (idSale, date sale, date livraison, etat livraison, client, List products, totalPrice) : Buy as a Guest (send mail), Buy after sign in

Menu

Pagination (or load more) + verification fields

Suggestions

Refactoring + good practice + separate admin/client interfaces

Internationalization

Contact us, legacy, footer

CSS

Testing

To correct and optimize:

Interceptor authentication

Bug access to controller with deleted token

Bug display of categories on every screen

Redirect after login

Add image

Filter according to each category of product

The website should be server-side rendering

Add reference: gender men women bug

Add reference: when choosing a category you should change the select to be added

CSRF is enabled by default in the server side (java), we'll need to include the CSRF token in our requests on the client side as well: <input type="hidden" name**="${\_csrf.parameterName}"** value**="${\_csrf.token}"**/>

Filter MEN-WOMEN -> Category -> products Men (for this category)