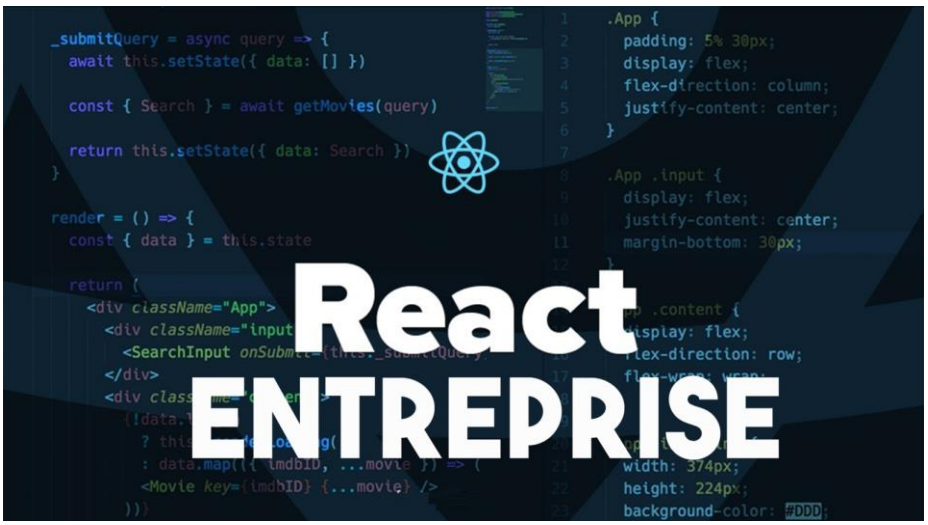
FORMATION



Fiches récapitulatives

Mike Codeur

Site : https://formations.mikecodeur.com

E-mail support@mikecodeur.com

Objectif de ce document

L'objectif de ce document est de faire un récapitulatif des concepts les plus importants pour maitriser le Framework React JS.

Il fait référence à de nombreux liens externes et vidéos pour en apprendre d'avantage (Doc JS, POO etc)

Table des matières

Module 2 – Rappel JS OBLIGATOIRE

Objectif du module

L'objectif de ce module est de montrer les prérequis en HTML et JS pour continuer cette formation.

Le but de ce module n'est pas de t'apprendre tous le HTML JS, mais de faire un récapitulatif des choses essentielles. De nombreux liens vidéos sont présent, il ne faut pas hésiter à les consulter et pratiquer.

Il est fortement conseillé de manipuler correctement tous ces concepts avant de passer à la suite.

A retenir

Le minium d'un page HTML

Tout site WEB est composé d'une structure HTML qui contient des balises.

Les balises sont imbriquées (contiennent forcement une ouverture <balise> et une fermeture </balise>)

Voir cette vidéo : https://www.youtube.com/watch?v=Tg1xZic-JQs&t=140s

Au minimum une page HTML ressemblera à ça

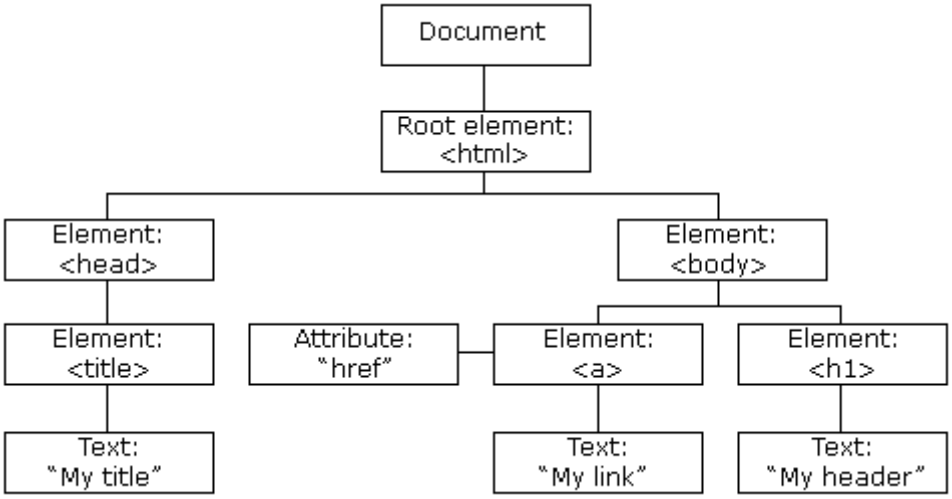
```
<!DOCTYPE html>
<html>
  <head>
  </head>

  <body>
  </body>
</html>
```

LE DOM

Pour simplifier, le DOM (Document Object Model) est la représentation de la structure de la page HTML sous forme d'objet.

Le but étant de pouvoir modifier cette structure dynamiquement via JavaScript.



Exemple de DOM

JavaScript

Pour utiliser JavaScript dans une page HTML il y a 2 possibilités :

- Soit directement dans la page HTML avec la balise `<script>`

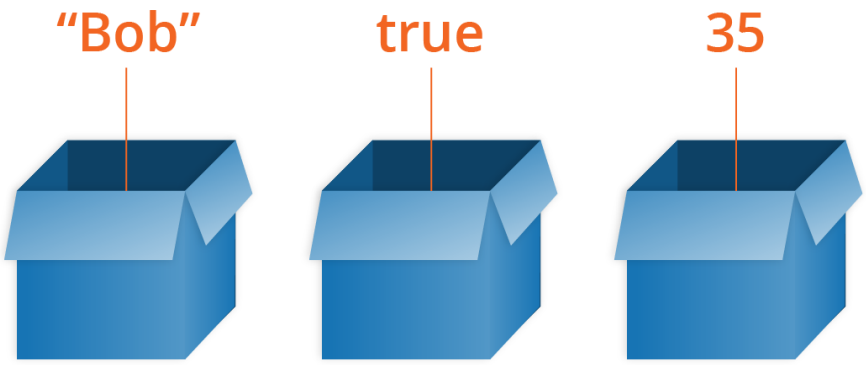
```
<body>
  <script type='text/javascript'>
      //a partir de là je peux utiliser du JS
      var name = "Mike";
      console.log("name " + name)
      //fin JS
  </script>
</body>
```

- Soit en créant du code JS dans un fichier a part (exemple `monScript.js`) et en l'incluant

```
<head>
     <script src="monScript.js"></script>
</head>
```

JavaScript (les variables)

Une variable est une sorte de boite, un espace, ou l'on y stocke des données.



Exemple de 3 variables

Une variable possède un nom unique.

Pour déclarer une variable on utilise le mot `var` suivi du nom de la variable `var age`

Pour remplir (affecter) des données dans une variable ou utilise `=`

```
var age = 21;    //ici a met la valeur 21 dans la variable age
```

A NE SUTOUT PAS CONFONDRE AVEC == (qui permet de comparer)

JavaScript (les types de variables)

Les variables contiennent des données qui sont « typés »

Chaine de caractères, nombre, boolean (true/false) etc…

Voir cette vidéo : https://www.youtube.com/watch?v=yX3KvpkOgTs

```
var name = "Mike";
var age = 21;
var book = [];
var book = {};
```

JavaScript (les fonctions)

Les fonctions font partie des briques fondamentales de JavaScript.

Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur.

Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Fonctions

Voir la vidéo : https://www.youtube.com/watch?v=QcJi-Tiie_s

```
//declaration de la fonction
function maFonction(ageparam){
     var chaine = "age " + ageparam;
      return chaine;
 }
//appel de la fonction
 var a = maFonction(20);
 console.log("a",a); //log
```

JavaScript (les Arrays)

Les Arrays (tableaux), **qu'il ne faut surtout pas confondre avec des tableaux HTML,** sont des variables qui contiennent d'autres variables :

C'est variables sont stockées à un index 0, 1 , 2 , 3 etc …

Dans cette exemple je mets 4 éléments dans mon tableau.

```
var monArray = [];
monArray.push("a"); //sera stocker à l'index 0
monArray.push("b"); //sera stocker à l'index 1
monArray.push("c"); //sera stocker à l'index 2
monArray.push("d"); //sera stocker à l'index 3
console.log(monArray);
```

Pour y accéder on fera.

```
monArray[0]; //retourne "a"
monArray[1]; //retourne "b"
monArray[2]; //retourne "c"
monArray[3]; //retourne "d"
```

Voir la vidéo : https://www.youtube.com/watch?v=mFdiyaTKr6w

JavaScript (les Objects)

Les Objects en Javascript sont très courant, il est important de bien les maitriser.

Contrairement aux Arrays qui sont un couple de **index/valeur**

Les Objects sont un couple clef/valeur (exemple **nom/mike** et **age/50**)

```
var monObj = {
     nom : "mike",
     age : 50
 };
```

Voir la vidéo : https://www.youtube.com/watch?v=Rk7uQSVIRek

JavaScript (les Opérateurs de comparaisons et les Conditions)

Dans tous les langages de programmation on a besoin de comparer des variables.

Exemple de la variable

```
var age = 21;
```

On veut conditionner l'exécution de partie de code en fonction de la valeur de age.

Exemple :

```
if (age > 18) {
  //je fais ceci uniquement si age > 18
} else {
  //sinon je fais cela
}
```

Le signe > est un operateurs mais il en existe beaucoup d'autre (voir tableau ci-dessous)

Opérateur	Correspond
==	égal à
!=	différent de
===	contenu et type égal à
!==	contenu ou type différent de
>	supérieur à
>=	supérieur ou égal à
<	inférieur à
<=	inférieur ou égal à

Voir la vidéo sur les opérateurs : https://www.youtube.com/watch?v=fZInsylkNto

Les conditions If Else : https://www.youtube.com/watch?v=09NPM2sTYwg

Module 3 – Philosophie de React JS

Objectif du module

L'objectif de ce module est te montrer la philosophie de React JS.

A retenir

Avant REACT

Il y avait quelques librairie JS comme JQUERY qui permettait de faire des choses sympas en Front mais ce n'était pas industrialisable (c'est à dire utilisable sur de gros projets)

Les problèmes

- Beaucoup de duplication / peu modulaire / peu paramétrable
- Peu industrialisable / maintenance énorme
- SITE FULL JS ? Même pas en rêve

Avec REACT

Avec l'arrivé des Framework Front end comme React. Il est désormais possible de construire un site complet en Full Javascript.

Avec le même niveau de qualité que les sites fait avec des langages backs (PHP JAVA etc)

Les avantages de React

- Approche orientée composants
- Composants réutilisables et paramétrables
- Composant = simple function
- Simplicité (Render div)
- SPA (Single page application)
- Composant avec ETAT
- JS standard
- Ultra RAPIDE
- Code compréhensible (Grace a JSX)
- Grosse communauté
- Les Tests Unitaires

Module 4 – Les erreurs de débutants

Objectif du module

L'objectif de ce module est de te montrer les erreurs classiques de débutants.

Il y a beaucoup de notion que tu ne comprendras pas tout de suite car ont les aborde dans les modules suivants.

Il faudra donc régulièrement revenir sur ce module au fur et à mesure de l'avancée de la formation.

A retenir

Les erreurs

- Multi react.Render

Dans la suite de la formation on parlera de

```
ReactDOM.render()
```

Mais je préfère le mettre directement dans les erreurs de débutants !

Il n'y a qu'un SEUL Render dans une application React ! LE code ci-dessous est interdit

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

//INTERDIT !!!
//d'en creer un 2ème

ReactDOM.render(
        <AutreComposant />,
        document.getElementById('root1')
      );
```

- Ne pas utiliser ES6 / fonctions fléchées / ternaires

Il est important de maitriser les fonctions fléchées le plus rapidement possible et de les utiliser (module 7)

- Ne pas assez découper les composants

Beaucoup de débutants créent un gros composant avec une Div et tout le HTML dedans. React permet un découpage très fin, il ne faut pas s'en priver !

- Confondre state et props

2 principes des bases de React JS qu'il ne faut surtout pas confondre (on les abordera dans les modules suivants)

- Faire trop de composants (avec état) vs pure fonction (sans état)

Nous verrons dans les modules suivants les 2 manières de créer des composants. Beaucoup de débutants créer des composant avec état alors que ce n'est pas nécessaire.

- Ne pas faire des composants génériques

Un composant générique est un composant qui peut faire plusieurs choses.

Exemple : Au lieu de faire un composant qui affiche la date, un composant qui affiche l'heure.

Il est possible de faire un Composant qui affiche soit la date, soit l'heure.

Une erreur consiste à créer trop de composant au lieu de regrouper 2 composants en 1.

- Faire des composants trop génériques

Ici c'est l'erreur inverse. Parfois les débutants veulent faire un composant trop générique. C'est-à-dire qu'il fait trop de choses en 1.

Exemple un composant qui affiche la date, l'heure, le nom du user, son nombre de messages. Bref un composant fourre-tout.

Il faut éviter cela

Module 5 – React JS en 5 minutes

Objectif du module

L'objectif de ce module est de te montrer rapidement React JS.

En règle générale React JS s'utilise avec de nombreux outils (Babel, Node, ES6 Webpack etc …) et les débutants s'y perdent.

Le but de ce module est de te montrer React JS seul dans une simple page HTML.

L'idée est de te montrer son fonctionnement comme une simple libraire.

Ce type d'usage n'est pas recommandé sur les vrais projets

A retenir

Le minimum pour utiliser React JS

- La librairie React et React DOM

  Il s'agit des 2 libraires nécessaires pour faire fonctionner React.

  Sans ces 2 librairies il n'est pas possible d'utiliser React et ses fonctionnalités.

- La librairie Babel

  Qui permet d'utiliser le JS standard ES6

  On en reparlera dans le module 7

Nous n'allons pas télécharger ces librairies en local, nous allons les utiliser directement sur un serveur distant en les incluant dans notre page HTML :

```
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
```

A partir de maintenant nous pouvons utiliser React, Créer des composants dans notre page HTML (ou dans un fichier JS à part)

La div principale

Toute application React requiert une div principale vide dans laquelle seront injectés les composants React

Dans notre cas nous allons lui donner l'id root

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

Le ReactDOM.Render()

C'est une fonctionnalité de React qui permet d'injecter un ou plusieurs composants dans la div principale de notre application

Exemple

```
ReactDOM.render(
        <Composant />
        document.getElementById("root")
    );
```

Un composant (Function)

Pour créer un composant en React, il suffit de créer une simple fonction qui retourne du code HTML

Par exemple :

```
function ComposentSimpleH6(){
        return (
            <div>
                <h6>pure compo h6</h6>
            </div>
        );
    }
```

Un composant avec PROPS

Un props (propriétés) sont des variables que l'on peut passer à un composant

Exemple de composant avec 2 props : label et age

```
function HelloWord({label ,age}){
        return (
```

```
            <div>
                <h6>Hello {label} tu as {age} ans</h6>
            </div>
        );
    }
```

Utilisation du Composant

```
<HelloWord label={"Mike"} age="21"/>

<HelloWord label={"John"} age="25"/>
```

Un composant déclarer avec une Classe (ES6)

Ceci est une autre manière de déclarer un composant

```
class MonCompoMike extends React.Component {
        render(){
            return (
                <div>
                    hello {this.props.label} {this.props.age}
                </div>
            )
        }
    }
```

Imbrication de composants

Il est possible d'imbriquer les composants entre eux.

Exemple (`ComposentSimpleH6` dans `HelloWord`)

```
function HelloWord({label ,age}){
        return (
            <div>
                <h6>Hello {label} tu as {age} ans</h6>
                <ComposentSimpleH6> </ComposentSimpleH6>
            </div>
        );
```

```
    }
```

Module 6 – JSX

Objectif du module

L'objectif de ce module est de te faire découvrir JSX

A retenir

Avant JSX

Avec JavaScript il est possible de manipuler le DOM.

Par exemple créer des éléments HTML avec du code JavaScript.

Prenons l'exemple de la création d'une balise

```
<h1>Hello World</h1>
```

En JavaScript il faudrait faire

```
<script type='text/javascript'>
        var h = document.createElement('h1');
        var t = document.createTextNode('Hello world');
        h.appendChild(t);
        document.body.appendChild(h);
</script>
```

Avec JSX (requiert Babel)

Avec JSX tu peux créer des variables qui représentent des éléments HTML directement dans le code JS

Ces 4 lignes de code au-dessus se transforment en

```
const elt = <h1>Hello World</h1>;
```

Du coup l'utilisation dans React y est simplifiée

```
function CompoSimple(){
      return (
           <div>{elt}</div>
       )
}
```

Render Avant JSX

Sans JSX il n'est pas possible de manipuler les composants React comme des balises HTML.

```
const e = React.createElement;

ReactDOM.render(

  e(ComponentSimple),

  document.getElementById('root')

);
```

Render avec JSX

Avec JSX nos composants React s'utilisent comme des balises HTML `<ComponentSimple />`,

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

Module 7 – ECMAScript 6

Objectif du module

L'objectif de ce module est de te faire découvrir le nouveaux Standard de JavaScript ES6

A retenir

ECMAScript (ES)

ECMAScript est le standard de Javascript.

La première version date de 1997

La grosse évolution fut la version 6 (ES6) qui apporte de nombreuses nouveauté

Qui supporte ES6, ES7, ES8 ?

Tous les navigateurs (Chrome, Safari, IE, etc) ne supportent pas toujours toutes les dernieres version de ES

Pour vérifier il existe un site

https://caniuse.com/#search=es6

Comment on fait pour être supporter partout ?

Il existe des outils qui vont permettent de coder en ES6.

Et de transformer ce code en code supporter par tous les navigateurs (Transpilation)

Le plus connu est Babel

A retenir de ES6

- Déclaration de variables (let vs const)

En ES6 on utilise plus le mot clef var pour déclarer une variable. A la place on utilise let et const

```
let name = "mike";
name = "Tony";

const KEY = "DFLKDKGLDFKGLD";
KEY = "nouvellekey";    ← INTERDIT, VA GENERER UNE ERREUR
```

Avec `let` on peut réaffecter (modifier) la variable (name passe de mike à Tony)

Avec `const` on ne peut pas réaffecter la variable.

- Déclaration de classes (Mot clef « Class » extends)

(Il est important de se renseigner sur la POO programmation orientée objet pour comprendre le concept de classe et héritage)

Video POO JS : https://www.youtube.com/watch?v=KjXBPJZPmYU

Avec ES6 Il est possible de déclarer des classes avec le mot clef `class`

Et faire de l'héritage avec le mot clef `extends`

```
class Vehicule {
    constructor(model,annee){
        this.model = model;
        this.annee = annee;
    }
}

class Voiture extends Vehicule {
    constructor(model,annee,color){
       super(model,annee);
       this.color = color;
    }
    updateAnnee(annee){
        this.annee = annee;
    }
}
```

Instanciation de la Classe

```
const maVoiture = new Voiture("twingo", 2015, "bleue")
```

- Paramètres par défaut

Il est possible avec ES6 de mettre des valeurs par défaut aux paramètre de fonction

```
function maFonction(x, vistesslumiere = 300000){
      console.log("x vaut" + x + " vistesslumiere vaut " + vistesslumiere)
 }

maFonction(6,1000); // vistesslumiere vaut 1000

maFonction(6); // vistesslumiere vaut 300000 (defaut)
```

- Modules (import export / export default)

Avec ES6 il est possible de créer des modules

Ce mécanise fonctionne avec les mots clefs export et import

Prenons un exemple de module qui contient des fonctions mathématiques (math.js)

Dans ce module la fonction `multiplier` et la constante `pi` pourrons être importés dans un autre module car on les a exporter (mot clef `export`)

La fonction `uneAutreFunction` ne sera visible et utilisable que dans le fichier math.js (car elle n'a pas été exportée)

```
export function multiplier(x,y){
    return x*y;
}

function uneAutreFunction(x,y){
    return x+y;
}

export const pi = 3.14;
```

Pour importer ensuite dans un autre module il existe 2 méthodes

La méthode ou l'on importe tout

```
import * as math from 'math';
console.log(math.multiplier(3x2))
console.log("pi vaut " + math.pi)
```

La méthode ou l'on importe unitairement

```
import {multiplier,pi} from 'math';
console.log(multiplier(3x2))
console.log("pi vaut " + pi)*/
```

- Arrow (fonction fléchée)

Les fonctions fléchée (arrow) sont une énorme avancée qui permettent de simplifier le code

Déclaration classique

```
function multiplier(x,y){
      console.log("multiplier " + x*y)
}

multiplier(3,6)
```

Fonction fléchée

```
let mul = (x,y) => {console.log("multiplier " + x*y)}
mul(3,6)
```

Module 8 – Webpack

Objectif du module

L'objectif de ce module est de te faire découvrir Webpack. Il n'est pas nécessaire de comprendre Webpack en profondeur pour avancer sur cette formation.

A retenir

Les problèmes

- Projets de + en + complexes
- Dépendances / js / css / sass / png etc
- Comment transpiller ?
- Comment minifier

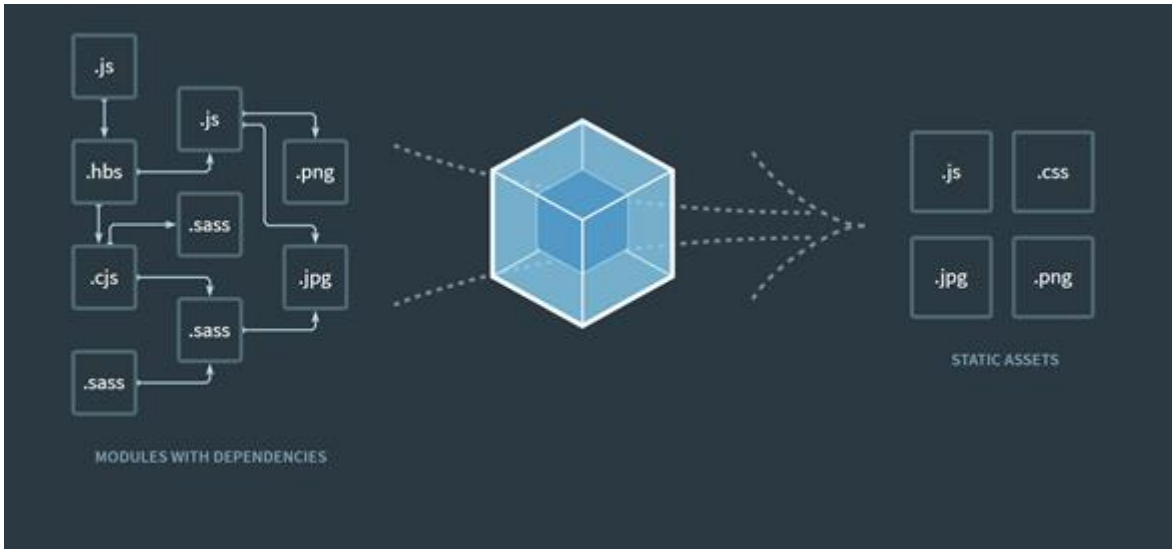
Les solutions

https://webpack.js.org/

- Webpack (gulp/ grunt / parcel)
- Principe : Code multi fichiers
- Compilation -> Bundle.js

Pour simplifier, tu codes dans plein de fichiers JS (des modules, des composants, des fichiers css, images etc)

Webpack va tout importer, compiler (babel) et générer un gros fichier avec tout dedans (bundle.js)



Module 9 – Les outils pour du JS avancé

Objectif du module

L'objectif de ce module est de te faire découvrir les outils pour du JavaScript avancé.

A retenir

Pour coder en React il te faut : Un IDE (éditeur de code), Node JS et NPM

Les éditeurs de code (IDE)

https://code.visualstudio.com/

Node JS

https://nodejs.org/en/

NPM (Node Package Manager)

https://www.npmjs.com/

NPM te permet de récupérer les dépendances de tes projets JS. React, Redux, Jest etc …

Quelques commandes à connaitre :

```
npm install

// permet d'installer (récupérer) toutes les dépendances

npm start

//permet de démarrer le projet
```

Module 10 – Initialisation du projet

Objectif du module

L'objectif de ce module est de créer un projet React JS de base avec create-react-app (utilitaire fourni par l'équipe React)

A retenir

La première chose à faire et de créer l'environnement du projet

Création d'un nouveau projet

Il faut exécuter `create-react-app` suivi du nom du projet

Exemple :

```
npx create-react-app mon-app

cd mon-app

npm start
```

Module 10B – Création de notre boite à outils

Objectif du module

L'objectif de ce module est de créer des composants React les plus courants et d'utiliser les mécanismes de bases de React

- Les composants pures (Simple fonction et sans état)
- ClassName pour affecter un style CSS
- Créer et utiliser des PROPS
- Les composants avec ETAT(State)
- Créer et utiliser des State (SET et GET)
- Jouer avec les Evènement React
- Data Binding (remonter (fonction en props))
- Ternaire (rendering conditionnelle)
- Boucler sur les components (map arrow)
- Cycle de vie

A retenir

Composant sans état

Pour créer un simple composant sans état (sans state) il suffit de créer une fonction JS qui retourne du code HTML.

Exemple de composant sans état qui retourne une balise <h6> et un texte

```
function PureCompo() {
    return (
        <div>
            <h6>Pure compo</h6>
        </div>
    )
}
```

On utilise ensuite ce composant comme une simple balise HTML de la manière suivante

```
//balise ouverture et fermeture
<PureCompo></PureCompo>

//ou en balise autofermante
<PureCompo />
```

Affecter un style (className)

Pour affecter un style CSS dans les composants React on utilise l'attribut `className` suivit du nom de la propriété css

```
export function PureCompo() {
    return (
        <div>
            <h6 className="pure-h6" >Pure compo</h6>
        </div>
    )
}
```

Composant avec propriétés (PROPS)

Tout composant React (avec ou sans état) peut avoir des props.

Les props sont des variables que l'on transmet aux *composants (parfois des fonctions que l'on passe en props*).

Tous les props sont passés dans un objet en paramètre de la fonction du composant.

Admettons un composant avec 2 props : `label` et `age`.

```
function PureCompoProps(props) {
    return (
        <div>
            <h6>Pure props compo {props.label} {props.age}</h6>
        </div>
    )
}
```

Par décomposition (ou descructuration) il est possible d'écrire le même composant comme cela :

```
function PureCompoProps({label, age}) {
    return (
        <div>
            <h6>Pure props compo {label} {age}</h6>
        </div>
    )
}
```

On utilise ensuite ce composant comme une simple balise **HTML qui possède des attributs** de la manière suivante :

```
<PureCompoProps label="john" age="50" />
<PureCompoProps label="julien" age="10" />
```

Composant avec état

Les composants avec états ne peuvent pas s'écrire avec de simple fonction pure *, il faut utiliser des Classes (voir module ES6 sur les classes)

Vidéo sur les classes : https://www.youtube.com/watch?v=KjXBPJZPmYU

** depuis React 16.8 et les hooks c'est désormais possible mais nous ne les aborderons pas ici*

Les props sont des variables figées, une fois renseignée **on ne peut pas les modifier** à l'intérieur du composant

Les states sont des variables qui peuvent évoluer (être modifier) dans le composant, lorsqu'un state change, l'affichage du composant est mis à jour

Exemple : Un composant avec un props label et un state nom

```
class StateCompo extends Component {
    constructor(props){
        super(props);
        this.state = {nom : ''};
    }

    render(){
        return (
```

```
        <div>
            <h6 className="state-h6">Compo State {this.props.label}</h6>
            <input type="text" value={this.state.nom}></input>
        </div>
        )
    }
}
```

L'utilisation se fait de la manière suivante (comme un sans état)

```
<StateCompo label="toto" />
```

Dans le constructeur (*le constructeur est la première chose exécutée dans une classe*) il est possible d'initialiser les states par défaut :

```
this.state = {nom : ''};
```

Pour utiliser un state dans le composant il suffit de faire `this.state.<nom_du_state>`

```
{this.state.nom}
```

Pour les props `this.props.<nom_du_props>`

```
{this.props.label}
```

Modifier un State

Pour modifier un état il est **STRICTEMENT INTERDIT** de faire :

```
this.state.nom ="nouveau nom"    ; // INTERDIT
```

Pour modifier un State il faut passer par une fonction setState

Exemple

```
this.setState({nom : "nouveau nom"});
```

Interagir avec les évènements JS

En JavaScript il existe des centaines d'évènements comme par exemple : un clic de souris (onclick), l'appuie sur un bouton du clavier, le changement d'un texte dans un champs (onchange) etc etc …

Une liste ici

https://www.w3schools.com/jsref/dom_obj_event.asp

Il est également possible d'utiliser ces évènements en React.

Nous allons dire d'exécuter une fonction lors du changement du texte dans un champs input.

Reprenons notre champs input du composant

```
<input type="text" value={this.state.nom}></input>
```

Nous allons y ajouter l'évènement onChange et lui dire d'appeler une fonction que l'on appellera `handleChange`

```
<input type="text" value={this.state.nom} onChange={this.handleChange}></input>
```

*Note : onChange envoi un évènement avec un paramètre **e** qui nous permet d'accéder au nouveau texte grâce à : **e.target.value***

```
handleChange(e){
   console.log("la nouvelle valeur est ", e.target.value )
}
```

En React on souhaite souvent mettre à jour notre state sur un évènement.

Comme vue plus haut pour mettre à jour le state il faut utiliser

```
this.setState({nom : "nouveau nom"});
```

Avec la valeur reçue de l'évent cela donnerait

```
handleChange(e){
        console.log("la nouvelle valeur est ", e.target.value )
        this.setState({nom : e.target.value });
}
```

CECI NE FONCTIONNERA PAS CAR LE MOT CLEF THIS NE SERA PAS RECONNU (BINDER)

Plus d'info sur `this` :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Op%C3%A9rateurs/L_op%C3%A9rateur_this

Pour résoudre ce problème on utilise BIND ou les fonctions fléchées

THIS, BIND et Fonction fléché

Pour pouvoir utiliser `this.setState` dans une fonction (`this` correspond au composant React et `setState` à une fonctionnalité de React) il y a 2 méthodes :

La première méthode consiste à **binder** la fonction (c'est-à-dire a lui spécifier qui est this) pour cela on utilise la fonction BIND dans le constructor

```
constructor(props){
        this.handleChange = this.handleChange.bind(this)
 }
```

Maintenant le `this.setState` fonctionnera dans notre fonction `handleChange`

```
handleChange(e){

        console.log("la nouvelle valeur est ", e.target.value )

        this.setState({nom : e.target.value });

}
```

La 2ème méthode consiste à utiliser une fonction fléchée. (Avec une fonction fléchée le this est automatiquement binder)

Pour cela plus besoin d'utiliser la bind dans le constructeur (`this.handleChange.bind(this)`)

Il est possible de définir la fonction handleChange de la manière suivante

```
handleChange = (e) => {
        this.setState({nom : e.target.value});
}
```

Remonter les events au Composant parent

Jusque-là notre évent est resté confiner à l'intérieur du composant React.

Dans certain cas il est nécessaire de faire remonter l'évent au composant parent.

Pour cela on utilise un technique simple : on utilise un props en fonction.

Comme on l'a vu plus haut les props sont des variables ou des functions.

Pour faire cela on va créer un props qui fait appel à une fonction

```
<StateCompo label={libelle} onInputChange={this.onInputChange}/>
```

```
onInputChange = (val) => {
    //code du compo parent
}
```

Et dans le composant `StateCompo` il ne reste plus qu'a appeler cette fonction avec comme paramètre la valeur à remonter

```
handleChange = (e) => {
        this.setState({nom : e.target.value});
        this.props.onInputChange(e.target.value);
}
```

Ternaire (rendering conditionnel)

Une expression ternaire est une sorte de IF ELSE en une seul ligne avec 3 termes

La conditions suivit de ? l'instruction si c'est ok : instruction si c'es KO

```
CONDITION ? OK : ELSE;
```

Prenons l'exemple suivant :

```
let text = '';
if (age > 18) {
    text = 'tu es majeur'
} else {
    text = 'tu es mineur'
}
```

Donne ça en ternaire

```
(age > 18) ? text = 'tu es majeur' : text = 'tu es mineur' ;
```

Les ternaires sont très utilisées en React pour condition le rendu.

Admettons 2 composants

```
<CompoMineur />
<CompoMajeur />
```

Un pour les mineurs, un pour les majeurs et que l'on veuille conditionner l'affichage au contenu de la variable age.

Il est possible d'utiliser une ternaire comme cela

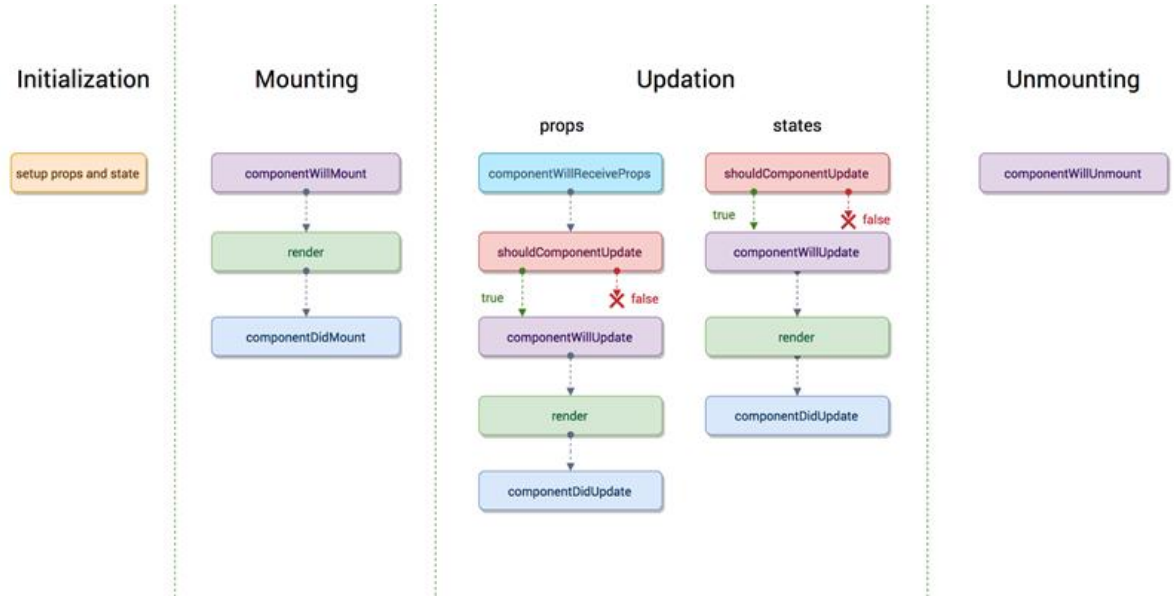
```
{ (age>18) ?   <CompoMajeur /> : <CompoMineur /> }
```

Le cycle de vie d'un composant avec Etat React

Les composants React ont un cycle de vie.

A chaque étape du cycle de vie il y a des fonctions que l'on peut utiliser (de manière optionnelle)

Ci-dessous toutes les étapes du cycle de vie.



En vert la fonction RENDER() est la fonction qui est appelé pour rafraîchir (mettre à jour) l'affichage.

Prenons un exemple concret d'utilisation

Admettons un composant avec un props sexe pouvant contenir 2 valeurs possibles

```
sexe="h"
```

ou

```
sexe="f"
```

Et que vous vouliez afficher un texte :

Bonjour vous êtes une homme ou *Bonjour vous êtes une femme*

Et voir même aller plus loin, mettre un fond bleu ou rose etc …

Avant d'afficher le render on a besoin de faire des opérations (calcul du texte, modification de la variable color etc …)

Pour cela dans la méthode avant le render, componentWillMount on va faire ces calculs

```
componentWillMount(){

        if (this.props.sexe === "h") {
            text = "Bonjour vous etes un homme"
            bgColor = "blue"
        } else {
            text = "Bonjour vous etes une femme"
            bgColor = "rose"
        }
        //etc etc autre code a executer avant le premier render
    }
```

Et il est possible d'exécuter du code à chaque étape du cycle de vie d'un composant. Uniquement en cas de besoin.

Toutes les fonctions possibles de cycle de vie sont ici :

```
class CompoCycleDeVie extends Component {
    constructor(props){
        super(props);
        console.log("Je passe dans le constructor");
    }
    componentWillMount(){
        console.log("Je passe dans le componentWillMount");
    }
    render(){

        console.log("Je passe dans le render");
```

```
        return (
        <div>
        </div>
        )
    }
    componentDidMount(){
        console.log("Je passe dans le componentDidMount");
    }
    componentWillReceiveProps(){
        console.log("Je passe dans le componentWillReceiveProps");
    }
    shouldComponentUpdate(){
        console.log("Je passe dans le shouldComponentUpdate");
    }
    componentWillUpdate(){
        console.log("Je passe dans le componentWillUpdate");
    }
    componentDidUpdate(){
        console.log("Je passe dans le componentWillUpdate");
    }

}


export default CompoCycleDeVie;
```