# Memo React

## Installation pour le développement (pas en PROD !)

* Importer les scripts de react et reactDOM depuis les liens CDN REACT, utiliser l’attribut defer !
* Pour utiliser le JSX il faut installer babel :

    <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>

    <script src="app.js" type="text/babel" defer></script>

## Ajouter un élément : sans JSX

const title = React.createElement("h1", { }, "Bonjour le monde") // créer un node H1 avec du texte

ReactDOM.render(title, document.querySelector("#main"))  //ajoute notre title au #main

L’intérêt de react comparer à un innerHTML est qu’il va chercher quel élément du node sera modifier pour ne modifier que celui-ci quand on fait des mis à jour.

## Ajouter un élément : avec JSX

const monTexte = <span id="monId"> ça va ? </span>

const title = <h1 className="cN"> Bonjour les gens {monTexte} </h1>

ReactDOM.render(title,document.body)

Le JSX ressemble beaucoup au HTML, cependant il y a des différences :

* Plutôt que d’écrire les attributs class on utilise className,
* On peut afficher nos variable JavaScript en JSX en les mettant entre accolade { }
* Il ne faut qu’une seule racine, si plusieurs englober avec <React.Frament> </React.Frament>

## Créer un composant réact

// créer un composant react : son nom commence TOUJOURS par une majuscule

class Welcome extends React.Component{

    constructor(props){

        super(props) // props est un objet qui contient les données transmis

        this.state = {date: new Date()} // les state sont les données qui vont évoluer

        this.timer = null

    }

    // determine quand un composant a été monté : créer un timer, on bind this pour ne pas le perdre

    componentDidMount () { this.timer = window.setInterval(this.tick.bind(this), 1000) }

    // determine quand un composant a été suprimmer : suprime le timer

    componentwillUnmount () { window.clearInterval(this.timer) }

    // changer l'état d'un composant : ici on modifie le state date

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

    //Méthode render() obligatoire. On recupere le name transmis, le children est le texte entre <></>

    render(){

        return  <div>

                    <h1>Bonjour {this.props.name} </h1>

                    <p>{this.props.children + this.state.date.toLocaleTimeString()}</p>

                </div>

    }

}

// on ajoute notre composant Welcome sous forme de <></>, on ajoute a props la valeur name qu'ont pourra recuperer

ReactDOM.render(<Welcome name="arthur"> il est : </Welcome>, main)

## Modifier une state dans un setState (**attention : asynchrone** …)

    // Si on doit utiliser state ou props dans un setState faire une fonction tel que, IMPORTANT

    tick(){

        this.setState(function(state,props){

            return {value: this.state.value+1}

        // si je veux utiliser une valeur de setState qui sois bien à jour faire tel que

        this.setState({ n: value }, () => this.giveMeState()

        })

    }

## Valeur par défaut d’une props

// On peut donner des valeurs par defaut à des props

Incrementer.defaultProps = {

    start: 0,

    step: 1

}

## Ajouter un évènement (syntheticEvent)

    // on peut ajouter un evenement avec une propriété onclick={}, ne pas oublier de bind le this

    render(){

    return <div>Valeur : {this.state.n} <button onClick={this.increment.bind(this)}>Incrementer</button> </div>

    }

    // le e représente l'evenement, qui sera une class que créer React

    increment(e){

        console.log(e)

        // on annule le comportement par defaut de l'action (a utiliser quasi tout le temps)

        e.preventDefault()

    }

*React ne créer pas directement un eventlistener mais une class, ce qui permet d’utiliser un listener que ce sois dans le DOM ou sur mobile (React native)*

## Utilisation de bind(this) dans un render appeler toute les x seconde (optimisation perf)

        // Quand on doit utiliser un .bind(this), declarer les dans le constructor() :

        this.start = this.start.bind(this)

        this.pause = this.pause.bind(this)

        // puis utiliser tel que :

        <button onClick={this.start}>play</button>

## Composant controlé (formulaire & react)

Les value sont contrôlé par un state, pour modifier ce state on devra utiliser un event onChange ex :

 <label htmlFor="nom">Mon nom</label> // on n’utilise pas for mais htmlFor en JSX, for etant un mot clé en js

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

<textarea id="textArea" name="textArea" value={this.state.textArea} onChange={this.handleChange}/>

<input type="checkbox" name="checked" checked={this.state.checked} onChange={this.handleChange}/>

<select value={ this.state.value} onChange={this.handleChange}>

<option value="grapefruit">Pamplemousse</option>

<option value="lime">Citron vert</option>

</select>

// pour gerer la selection de plusieurs valeur d'un select

<select multiple={true} value={['B', 'C']}>

*NB : en HTML la valeur d’un textArea est définie dans l’enfant (entre les balises) en JSX il est définie par value*

onChange appel une méthode, Ici on met à jour notre state de manière générique, [name] est une propriété calculés

    // on permet de modifier la valeur de notre state via onChange

    handleChange(e){

        const name = e.target.name // recupere le name

        const type = e.target.type // recupere le ntype de l’input

        const value = type === 'checkbox' ? e.target.checked : e.target.value // recupere la value

        this.setState({

            [name]: value

        })

    }

Attention, ici nos composant sont contrôlé par react, ce qui veut dire que chaque modification d’un input va entrainer une modification du state et donc render() sera appeler autant de fois, ce qui peut amener des problèmes de performances. Si on n’a pas besoin de tracké la valeur (comme par exemple si on veut dire en temps réel si le mot de passe est assez long, qu’il comporte une majuscule etc) on créera un champ non contrôlé.

## Composant non contrôlé

// pour utiliser un composant non controlé

<input type="text" id="nom" name="nom" />

// pour pouvoir assigné une valeur par défaut, mais ne pas controler le composant

<input type="text" id="nom" name="nom" defaultValue="le nom"/>

## La composition

La composition permet de créer des conponent qui « hérite » d’autre conponent, ce qui rend l’heritage classic inutile

function Button ({children, type, ...props}) {

    return <button className={"btn btn-" + type} {...props}>{children}</button>

}

// La composition vient se substituer à l'héritage

function PrimaryButton ({children, ...props}) {

    return <Button type="primary" {...props}>{children}</button>

}