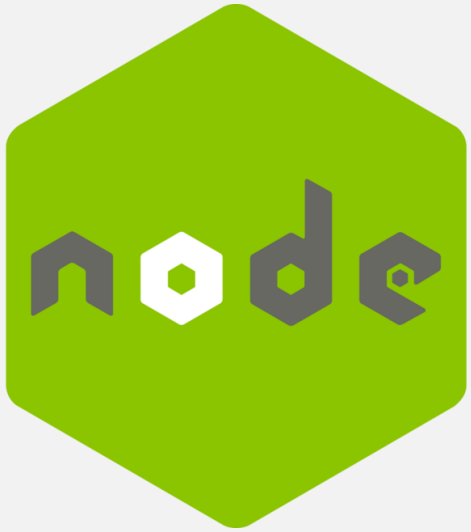
React JS

SOMMAIRE

ECMA Script 6

React

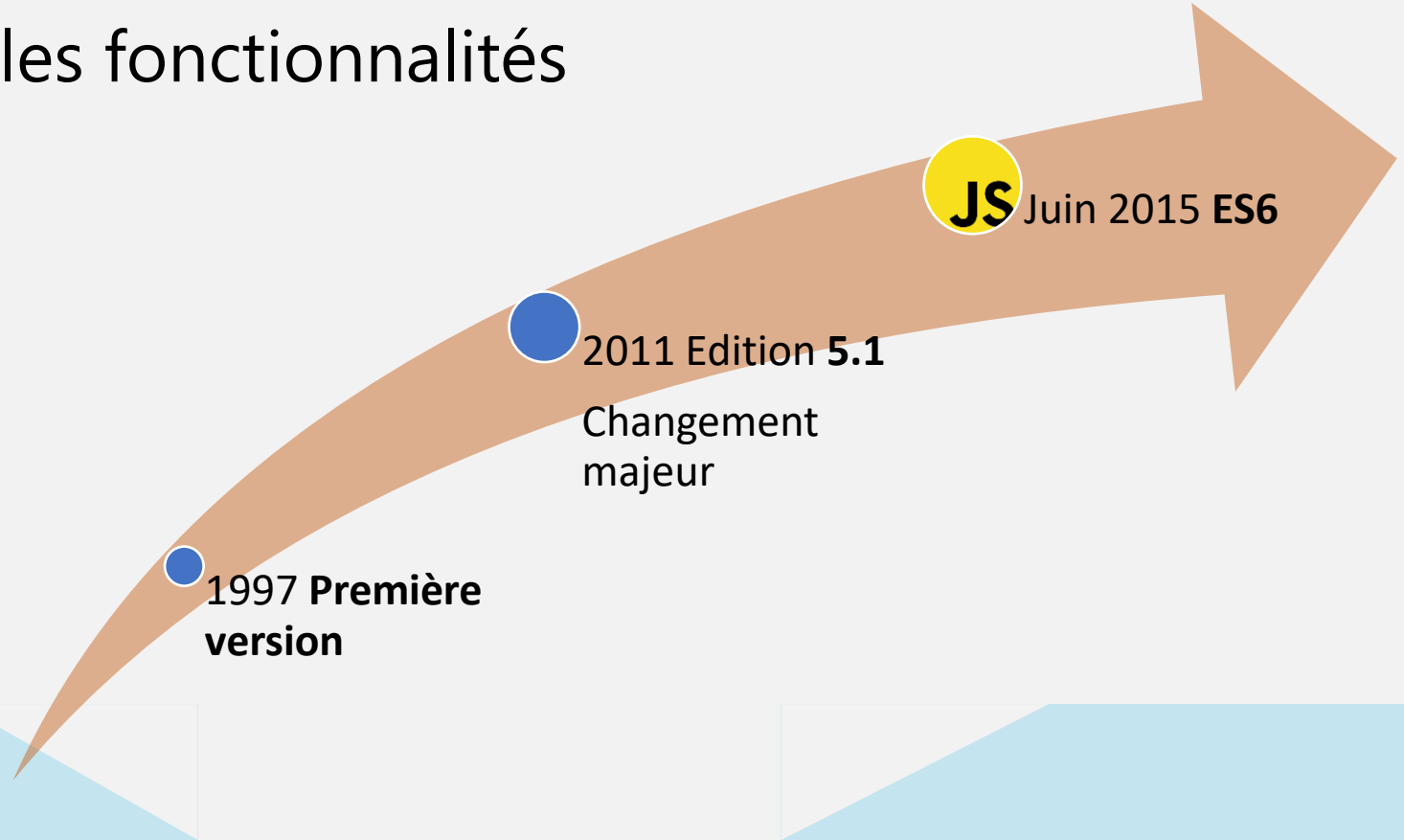
Avant de commencer



ECMA Script 6

Le JavaScript de demain, aujourd'hui

- Aussi appelé ES6 ou ECMA Script 2015
- Offre de nouvelles fonctionnalités



ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Constants

- Valeur immuable. Impossible de réassigner une nouvelle valeur.
- Utilisation du mot clé **const**

```
const iamConstante = "Variable
Immuable";
iamConstante = "problèmes";
```

Let

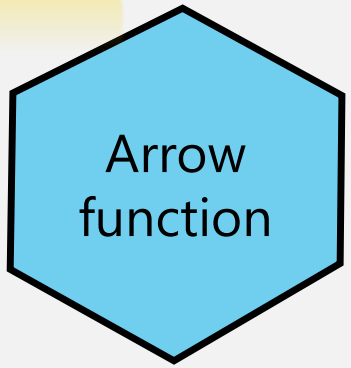
- Valeur muable. Peut être changé à tout moment
- Utilisation du mot clé **let**

```
let variabeTest = "Variable de
test";
variabeTest = "Pas de problème";
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui



- Définition de fonction plus courte.
- Ne redéfini pas le contexte courant (**this**).

{ } ES6

```
const firstFunction = value => value
+1;

const secondFunction = value =>
({value1: value, value2: value +1});

const thirdFunction = (value1,
value2) => value1 + value2;
```

{ } ES5

```
var firstFunctionEs5 = function (value) {
return value + 1; };

var secondFunctionEs5 =function (value) {
return { value1: value, value2: value + 1
}; };

var thirdFunctionEs5 = function (value1,
value2) { return value1 + value2; };
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Arrow function

{} ES6

```
this.nums.forEach((v) => {
if (v % 5 === 0)
this.fives.push(v)
})
```

{} ES5

```
var self = this;
this.nums.forEach(function (v) {
if (v % 5 === 0)
self.fives.push(v);
});
```

Le contexte parent est accessible dans une arrow function. Ce dernier n'est pas redéfini.

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Default parameter value

- Initialisation des paramètres dans la définition de la fonction.

{ } ES6

```
function functionTest (param1, param2
= 8, param3 = 16) {
      return x + y + z
}
f(1) === 25
```

{ } ES5

```
function functionTest (param1, param2,
param3) {
    if (y === undefined)
    y = 8;
    if (z === undefined)
    z = 16;
return x + y + z;
};
f(1) === 25;
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

- Opérateur de décomposition.

{ } ES6

```
const params = [ "hello", true, 7
]
const other = [ 1, 2, ...params ]
// [ 1, 2, "hello", true, 7 ]
```

{ } ES5

```
var params = [ "hello", true, 7
];
var other = [ 1, 2
].concat(params);
// [ 1, 2, "hello", true, 7 ]
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

String interpolation

- Concaténation de chaine simple ou multiple.
- Plus rapide et concis.

{ } ES6

```
const customer = { name: "Foo" };
const card = { amount: 7, product: "Bar",
unitprice: 42 };
const message = `Hello ${customer.name}`;
```

{ } ES5

```
var customer = { name: "Foo" };
var card = { amount: 7, product:
"Bar", unitprice: 42 };
var message = "Hello " +
customer.name ;
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Property Shorthand

- Syntaxe de création d'objet réduite et rapide.

{ } ES6

```
let x = 0, y = 0
obj = { x, y };
```

{ } ES5

```
var x = 0, y = 0;
obj = { x: x, y: y
};
```

ECMA Script 6

Le JavaScript de demain, aujourd'hui

- Ajout d'une propriété à la volée via le contexte courant.

{ } ES6

```
let obj = {
foo: "bar",
[ "baz"]: 42
}
```

{ } ES5

```
var obj = {
    foo: "bar"
};
obj[ "baz" ] = 42;
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Destructuring Assignment (**Array**)

- Assigner des variables provenant d'un objet ou tableau reposant sur leur structure.
- Très pratique pour la récupération de certaines variables ou constante par exemple.

{ } ES6

```
const list = [ 1, 2, 3 ]
const [ a, b , c ] = list;
```

{ } ES5

```
var list = [ 1, 2, 3 ];
var a = list[0], b = list[1], c= list[2]
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Destructuring Assignment (**Object**)

Destructuring Assignment (**Object**) ***deep**

- Assigner des variables provenant d'un objet ou tableau reposant sur leur structure.
- La récupération est similaire à la précédente.

{ } ES6

```
var { op: a, lhs: { op: b }, rhs: c } = getASTNode()
```

{ } ES5

```
var tmp = getASTNode();
var a = tmp.op;
var b = tmp.lhs.op;
var c = tmp.rhs;
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Modules (**Import / Export**)

- Permet de cloisonner des parcelles de code sous un espace de nom particulier.
- Importer ou exporter des valeurs ou composants nommés ou non.

- **Deux types** d'export :

Export nommé

- Utile pour exporter plusieurs valeurs.
- Obligation d'utiliser le même nom de l'objet exporté lors de l'importation

Export par défaut

- Utile pour exporter une seule valeur.
- Peut avoir n'importe quel nom lors de l'importation

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Modules (**Import / Export**)

{ } **Export** et **Import** nommé

```
export const myFunctionToExport = ()=> {};
import {myFunctionToExport} from
'./myFile';
```

{ } **Export** et **Import** par défaut

```
export default function (){};
import myfunction from './myfile';
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Modules (**Import / Export**)

- Importer **l'intégralité** d'un module.

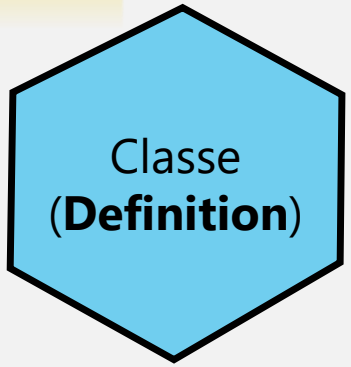
```
import * as myModule from
'/myModule.js';
```

- Importer un objet exporté avec un **alias.**

```
import {nameOfObjectIReallyReallyBig as shortName} from '/myModule.js';
```

ECMA Script 6

Le JavaScript de demain, aujourd'hui



- **Sucre syntaxique*** plus pratique et lisible permettant de définir une classe comme en POO. Utilisation de « **class** ».

**Les sucres syntaxique sont des expressions permettant de faciliter la vie du développeur. Utilisé dans un langage de programmation.*

{ } ES6

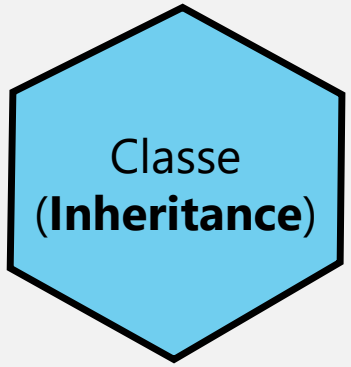
```
class User {
constructor (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.printUser(firstName, lastName);
}

printUser (firstName, lastName) {
  console.log(`Hello my name is
${firstName} ${lastName}`);
        }
}
```

{ } ES5

```
var User = function(firstName, lastName){
this.firstName = firstName;
this.lastName = lastName;
}

User.prototype.printUser = function(){
console.log('Hello my name is'+ this.firstName + ' '
+ this.lastName);
};
```

ECMA Script 6

Le JavaScript de demain, aujourd'hui



- Utilisation du sucre « **extends** ».

{ } ES6

```
class Human {
  constructor(age){
    this.age = age;
  }
}
class User extends Human {.....
```

{ } ES5

```
var User = function(firstName, lastName, age){
   this.firstName = firstName;
   this.lastName = lastName;
   Human.call(this, age);
}
User.prototype =
Object.create(Human.prototype);
User.prototype.constructor = User;
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Enhanced Object Literrals

- Permet de gagner du temps lors de la création d'un objet.
- Evite la lourdeur de réécriture des propriétés de l'objet.

{ } Création d'un objet rapide en ES6

```
const a=1, b= 2, c=3;
const imFlyObject = {
    a,
    b,
    c
};
```

{ } Création d'un objet rapide en ES5

```
const a=1, b= 2,
c=3;
const imFlyObject =
{
    a: a,
    b: b,
    c: c
};
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

Enhanced Object Literrals

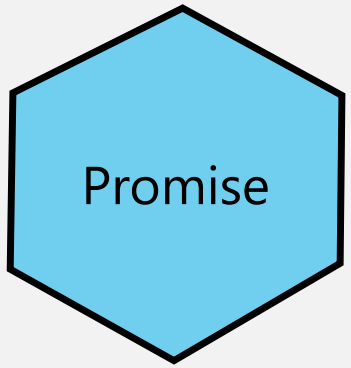
{ } Idem avec les fonctions !!

```
const lib = (() => {

function sum(a, b) { return a + b; }
function mult(a, b) { return a * b; }
    return {
        sum,
        mult
    };
})();
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui



- Objet permettant la gestion des traitements asynchrones.
- Une « promesse », représente une valeur qui peut être disponible, maintenant, dans le futur, voir jamais.
- Utilisation de l'objet **Promise**

```
const myFirstPromise = new Promise((resolve,
reject)=>{
//Si la promesse réussi
//resolve(valeur de retour)

//Si la promesse est rejetée
//reject(valeur de retour)
});
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

- Une fonction peut retourner une promesse. C'est même une très bonne pratique
- Utilisation de **then()** en cas de réussite et **catch()** en cas d'échec

```
const myFirstPromiseReturnFunction = ()=>
{
return new Promise((resolve, reject)=>{
// resolve ou reject ....
});
}
myFirstPromiseReturnFunction.then((value)=
>{

}).catch(()=>"Promesse rompue");
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui

- Possibilité de combiner plusieurs promesses.
- Il est possible d'attendre la fin de tous les traitements avant d'en exécuter un nouveau.
- Utilisation de Promise.**all**()

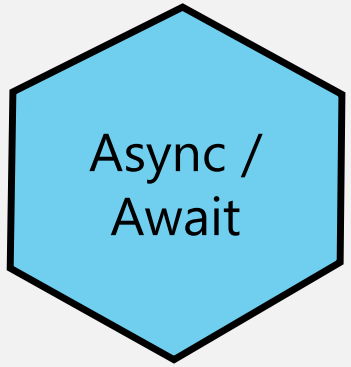
```
function fetchAsync (url, timeout, onData, onError) { … }

let fetchPromised = (url, timeout) => {
return new Promise((resolve, reject) => { fetchAsync(url,
timeout, resolve, reject) })
}
```

```
Promise.all(
[ fetchPromised("http://backend/foo.txt", 500),
fetchPromised("http://backend/bar.txt", 500),
fetchPromised("http://backend/baz.txt", 500) ])
.then((data) => {
let [ foo, bar, baz ] = data console.log(`success: foo=${foo}
bar=${bar} baz=${baz}`) }, (err) => { console.log(`error: ${err}`)
})
```

ES6

ECMA Script 6

Le JavaScript de demain, aujourd'hui



- Utilisation de ce sucre permettant la lecture et la gestion plus facile des méthodes asynchrones.

```
const functionAsync = async()
=>{

} ;

const callFunctionAsync = await
functionAsync();
```

ECMA Script 6

Le JavaScript de demain, aujourd'hui

❑ Résumé

- De nombreuses améliorations et confort du langage

- **Const** et **let** sont block scopés à la différence du var qui est fonction scopé. L'assignation fonctionne par référence et non par valeur.

- **Arrow function**, ne dispose pas de son propre contexte.

ECMA Script 6

Le JavaScript de demain, aujourd'hui

❑ Résumé

- **Default parameter**, utile pour passer des valeurs par défaut au paramètre de fonction.

- **Rest and spread operator**
  - ❑ Rest : Assemble plusieurs valeurs dans un tableau.
  - ❑ Spread : Eclater un tableau en une liste finie de valeurs.

- **Property Shortand** : pas obliger de réécrire les propriétés.

ECMA Script 6

Le JavaScript de demain, aujourd'hui

❑ Résumé

- **Destructuring** sur les tableaux, objet et nested objet
- **Module** : cloisonné le code.
- **Promise** : Gestion asynchrone.

ES6

A vous de jouer !

Avant de commencer

- Tous les développements doivent être effectués en mode "TDD"
- Utilisation du module **Jest** pour la mise en place des tests unitaires.

A vous de jouer !

{} **Etape 1 :** https://github.com/Gparquet/EcmaScript6Experience.git

{} **Etape 2 :** Aller dans le dossier puis taper **npm install**

{} **Etape 3 :** Dans le dossier **contact,** créer deux fichier - **contact.service.js. (**Ce dernier contiendra **toutes les méthodes permettant** de parser le fichier **contact.json).**

- **contact.service.spec.js. (**Contiendra tous les tests relatifs au fichier contact.service).

ES6

A vous de jouer !

{} Etape 4 : Enrichissement du service contact

- Création d'une méthode qui retourne les contacts par nom et prénom. Si l'utilisateur n'est pas trouvé renvoyer un message.

  - ❑ *Utilisation des arrow function*
  - ❑ *litteral expression*
  - ❑ *forEarch itértor, module*

A vous de jouer !

{} Etape 5 :

- Création d’une méthode qui retourne les contacts par leur identifiant et nom. Si l’utilisateur n’est pas trouvé renvoyer un message.

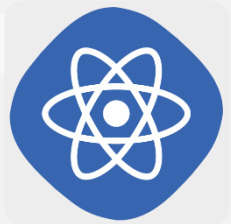
  - ❑ *Utilisation des arrow function*
  - ❑ *Litteral expression*
  - ❑ *forEarch itértor, module*

ES6

A vous de jouer !

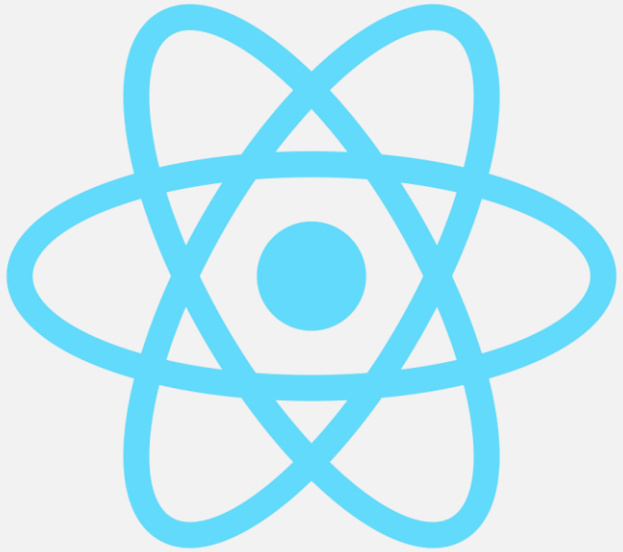
{} Etape 6 :

- Création d'une méthode permettant d'ajouter un nouveau contact.
  - ❑ *Utilisation des arrow function*
  - ❑ *litteral expression*
  - ❑ *Default parameter value*
  - ❑ *Destructuring, module*



ReactJS

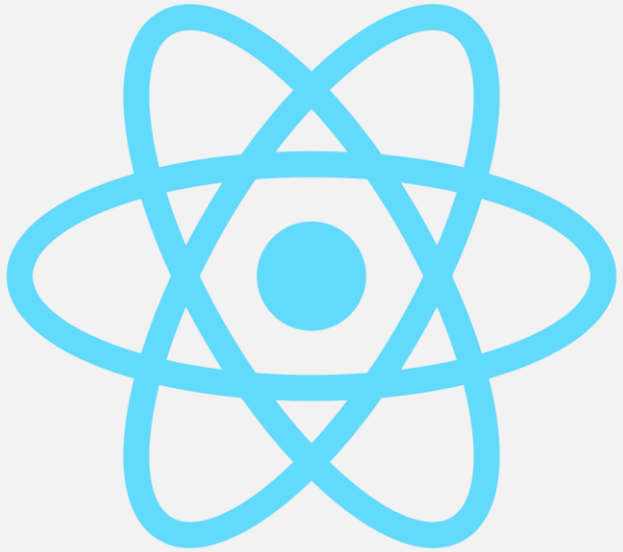
Vers un monde meilleur !



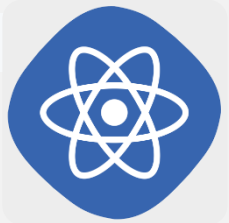
- Bibliothèque, non un Framework !
- Gestion uniquement de la vue (interface de l'application).
- Créé par Facebook depuis 2013.
- Pourquoi ????
  - Répondre au problème de l'application mono-page.

ReactJS

Vers un monde meilleur !



- ReactJS c'est aussi .....
  - ❑ Simplicité
  - ❑ Rapidité
  - ❑ Flexibilité
- Peut également être couplé avec un Framework tel que Angular par exemple.



ReactJS

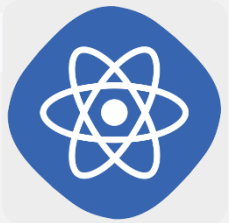
Vers un monde meilleur !

- ReactJS manipule un **DOM virtuel** et non celui du navigateur !!

- Tout est en composant en ReactJS. Ce dernier ne créer pas de HTML, mais une représentation sous forme d'objet et de nœuds de ce à quoi le HTML doit ressembler.

  - ❑ Pour rappel :

    - ✓ **DOM** = **D**ocument **O**bject **M**odel.
    - ✓ Interface entre le code et le HTML créé.
    - ✓ Représentation à un instant T la page visible par l'utilisateur.



ReactJS

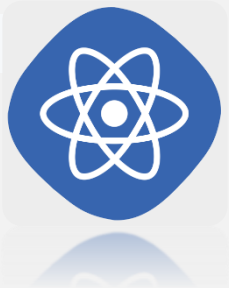
Vers un monde meilleur !

❑ Avant l'arrivée de ReactJS, les limitations :

- ✓ Performance réduite *(très visible sur les produits nomades)* lors d'une simple modification sur un DOM complexe.

- ✓ Suivre les changements d'état encore plus difficile. AngularJS en est l'exemple parfait avec sa méthode « ***Dirty Checking*** ».

  - ❑ Tous les objets sont surveillés en permanence, y compris ceux qui ne changent pas d'état.



ReactJS

Vers un monde meilleur !

- **DOM virtuel**

(virtual-dom)

❑ C'est quoi ?

- Outil permettant la représentation du DOM actuel complétement décorrélé de son homologue.
- Gestion des actions minimes à exécutés pour mixer les changements du DOM virtuel avec celui utilisé par le navigateur.

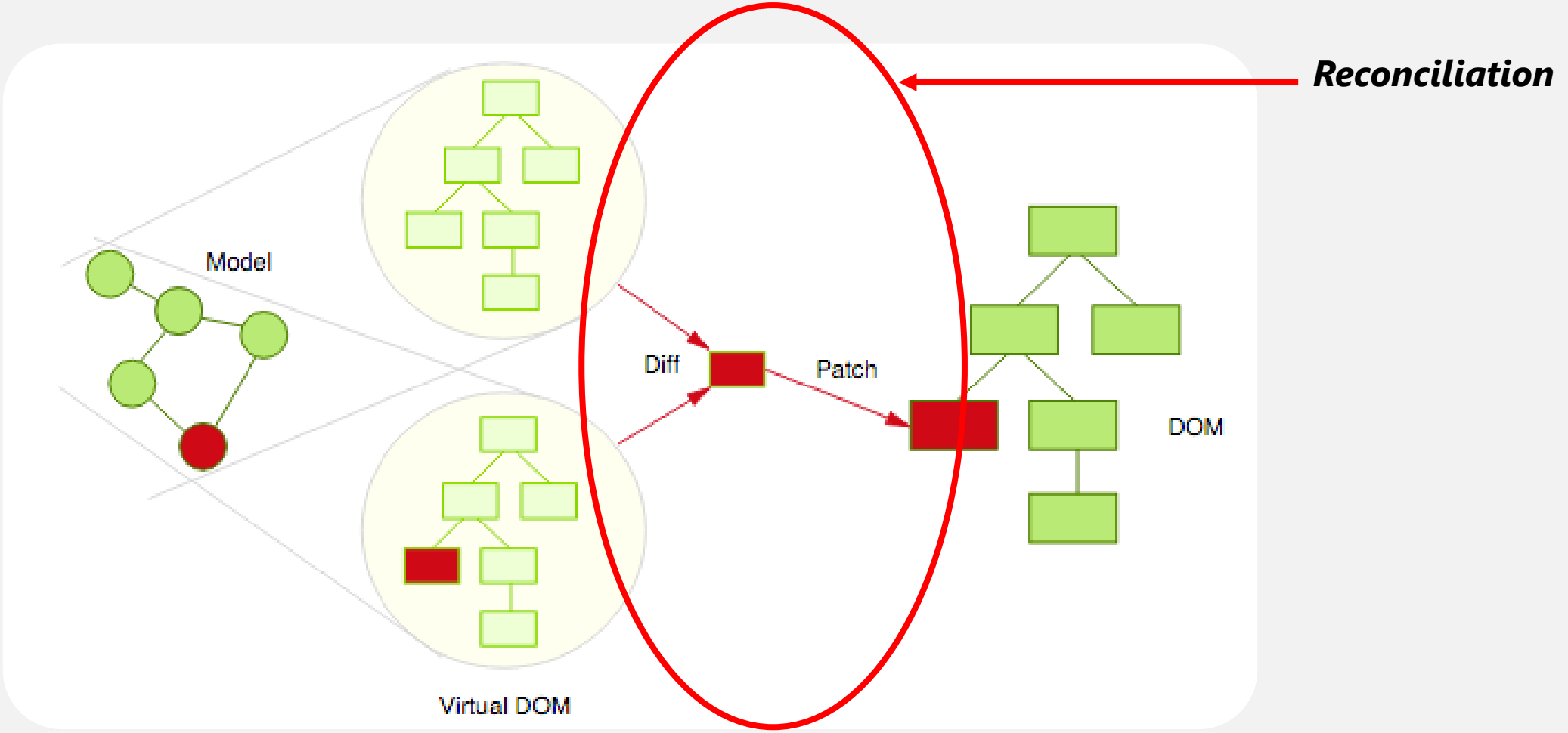
❑ Intérêt ?

✓ Gestion de la comparaison, modification du DOM coté JavaScript.

✓ Application plus rapide !

ReactJS

Vers un monde meilleur !

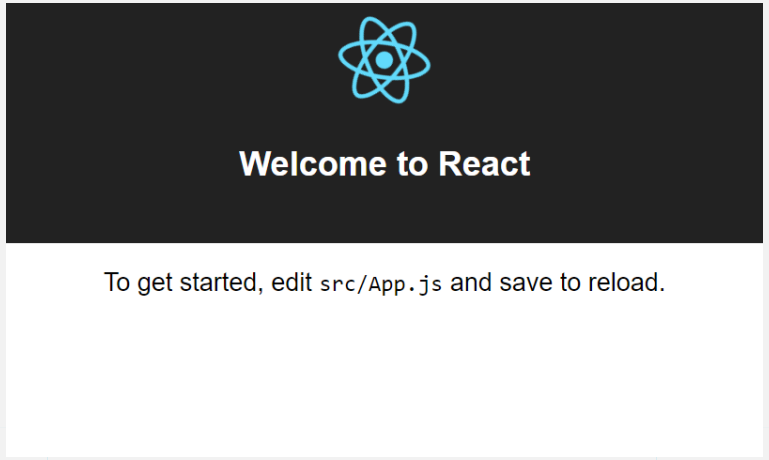


ReactJS

Vers un monde meilleur !

❑ Create-react-app

- Taper la commande **`npm init react-app my-app`**
- Aller dans le dossier puis faire **app-test** puis taper **npm start**



ReactJS

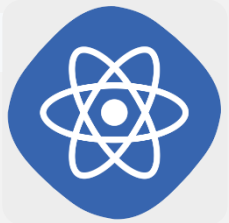
Vers un monde meilleur !

Public

- **Index.html** : page de démarrage de l'application

Src

- **App.js** : composant principal
- **Index.js** : point d'entrée principal pour le rendu des composants



ReactJS

Vers un monde meilleur !

❑ Create-react-app

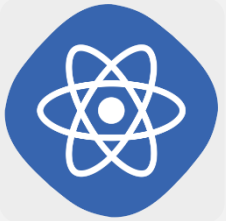
- C'est un module nodeJS permettant la création d'une application ReactJS disposant de toute la configuration nécessaire.

❖ Avantage

- Facile, rapide et assisté.

❖ Inconvénient

- Ignorance de la conception.



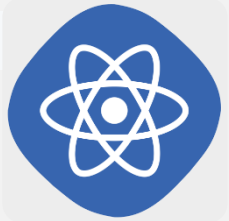
ReactJS

Vers un monde meilleur !

- Application de zéro …… Presque !

BABEL

ES6

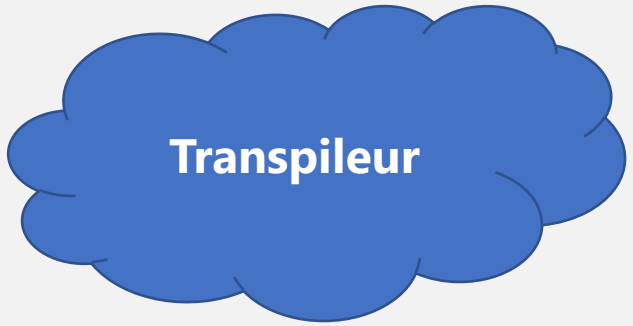


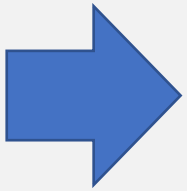
ReactJS

Vers un monde meilleur !

BABEL

```
(x) => x +1;
```

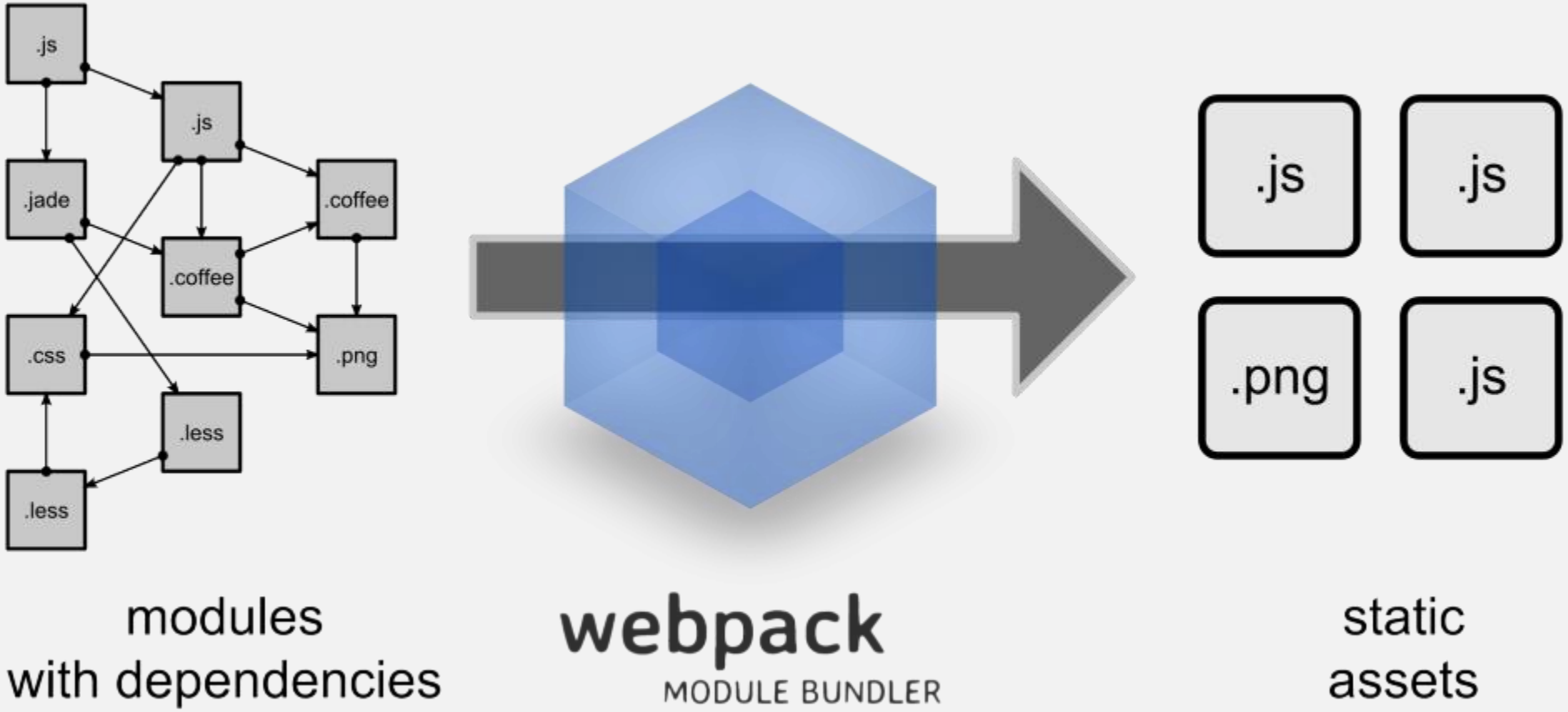


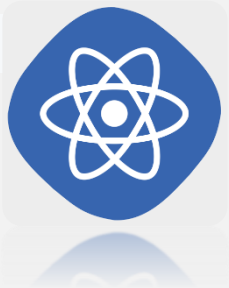


```
function(x){
      return x+1;
}
```

ReactJS

Vers un monde meilleur !

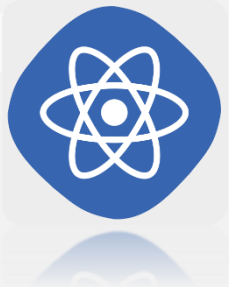




ReactJS

Vers un monde meilleur !

Orienté **Composant**
StateFull



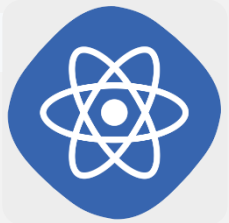
ReactJS

Vers un monde meilleur !

- Orienté composant

StateFull

StateLess

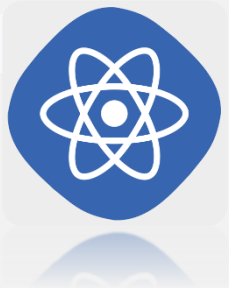


ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

- Qu'est ce que le principe de séparation de concept ??
- **ReactJS, ne fait pas de séparation de concept**. HTML et le JavaScript sont couplés dans un même composant.
- Un composant **peut contenir ou non** ses propres **états.**



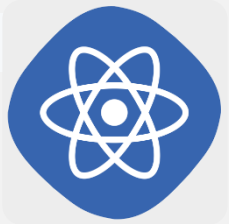
ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

- La première chose à importer lors de la création d'un composant

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



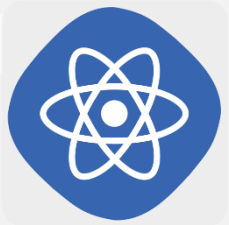
ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

❑**Exemple :**

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

class Person extends Component{
   render(){
      return <h1>Je suis un composant personne</h1>;
   }
}

export default Person;
```



ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

❑ Gestion des données

- Deux types de données en ReactJS (**props** and **state**).

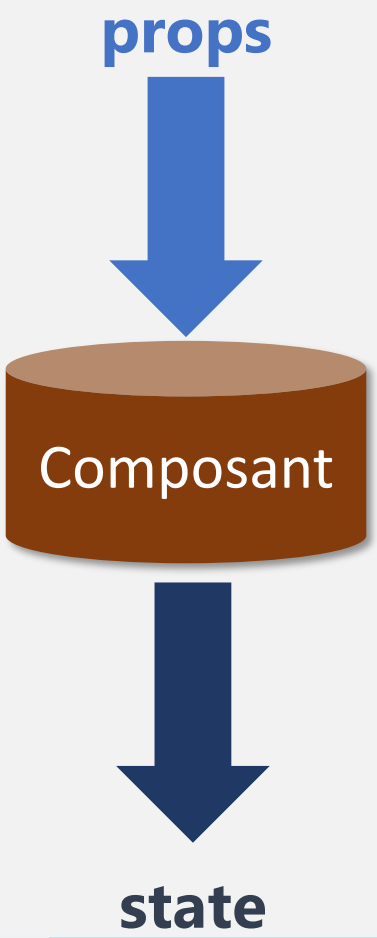
state	props
Données privées	**Données externes**

ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

Un composant peut changer ses données interne (***state***) mais pas ses données externe (***props***)



ReactJS

Vers un monde meilleur !

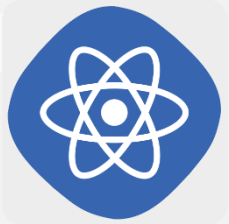
Orienté composant : *StateFull*

{ } Déclaration

```
class Contact extends Component{
render(){
    return
    <div id="container">
        <h1>Je suis un composant personne</h1>
            <p>
                Nom : {this.props.firstName}
                <br />
                Prenom : {this.props.lastName}
            </p>
    </div>
}
}
```

{ } Appel

```
<Contact
firstName="geoffrey"
lastName="parquet" />
```



ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

{ } Initialisation

- Tout ce passe via le **constructeur**

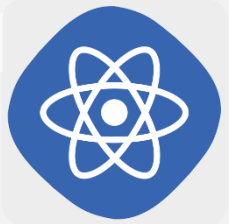
- Appel de la méthode **super()** permettant d'initialiser l'objet courant (**this**).

ReactJS

Vers un monde meilleur !

```
class Contact extends Component {
   constructor(){
      super();
      this.state = {
         firstName: '',
         lastName: ''
      };
   }……
```

```
render() {
   return
      <div id="container">
         <h1>Je suis un composant
         personne</h1>
            <p>
               Nom :
               {this.props.firstN
               ame}
               <br />
               Prenom :
               {this.props.lastNa
               me}
            </p>
</div>
```



ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

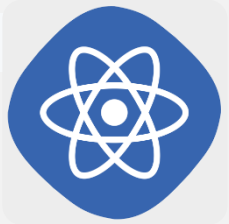
{ } Mise à jour du state

- **Bind** de **this** à la méthode. Si ce dernier n'est pas passé**, il ne peut pas être utilisé dans cette dernière.**

```
this.updateFirstName = this.updateFirstName.bind(this);
    updateFirstName(value){
        this.setState({
        ...this.state,
        firstName: value
    }):
}
```

- Utilisation de la méthode **setState**().

```
<button onClick={this.updateFirstName('Nouveau prénom')}></button>
```



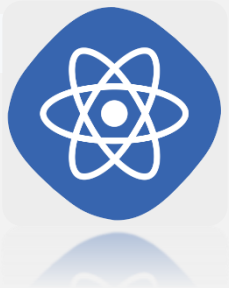
ReactJS

Vers un monde meilleur !

Orienté composant : *StateFull*

❑ Résumé

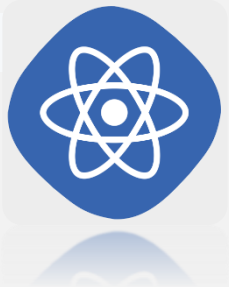
- Composant appelé ***StateFull***
- Utilisation de la classe ***Component*** de ***React***
- Deux types de propriété **state** et **props**
- State privé au composant, props extérieur au composant
- Modification du state par la méthode **setState()**
- Ne pas oublier de **binder this** au méthode si ce dernier doit être utilisé.



ReactJS

Vers un monde meilleur !

REACT-DOM



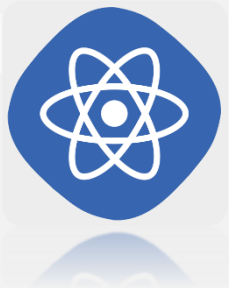
ReactJS

Vers un monde meilleur !

- **ReactDOM** s'assure de mettre à jour le DOM virtuel et le DOM physique

- Pour rendre un élément dans le DOM, il faut utiliser le méthode **render()** de **ReactDOM**

- Ce qui est initialisé dans la méthode, doit être la **racine de l'application**



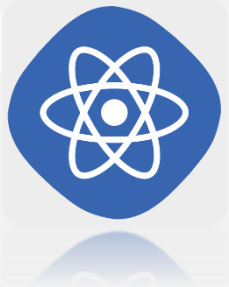
ReactJS

Vers un monde meilleur !

- **Exemple :**

```
const element = <h1>Hello, world</h1>;
ReactDOM.render(element,
document.getElementById('root'));
```

A vous de jouer !



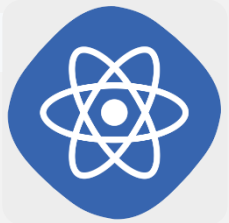
A vous de jouer !

{} Etape 1 : https://github.com/Gparquet/ReactExperience.git

{} Etape 2 : Placer vous dans le dossier, puis exécuter npm install

{} Etape 3 : Une fois terminé, exécuter la commande npm start

```
ERROR in multi (webpack)-dev-server/client?http://localhost:1988 (webpack)/hot/dev-server.js ./src/main.js
Module not found: Error: Can't resolve './src/main.js' in 'C:\GPA Projects\ReactExperience'
 @ multi (webpack)-dev-server/client?http://localhost:1988 (webpack)/hot/dev-server.js ./src/main.js app[2]
```

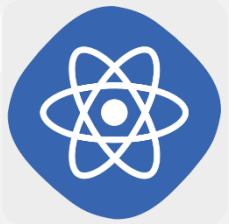


A vous de jouer !

{} Etape 4 : Ajouter un dossier **src**, et dans celui-ci, un fichier **main.js *(Point d'entré de l'application)*.**

{} Etape 5 : Créer un composant **App.js**, qui affiche « First Application React » en tant que titre principal

{} Etape 6 : Rendre le composant dans la page index.html en passant par le fichier main.js.

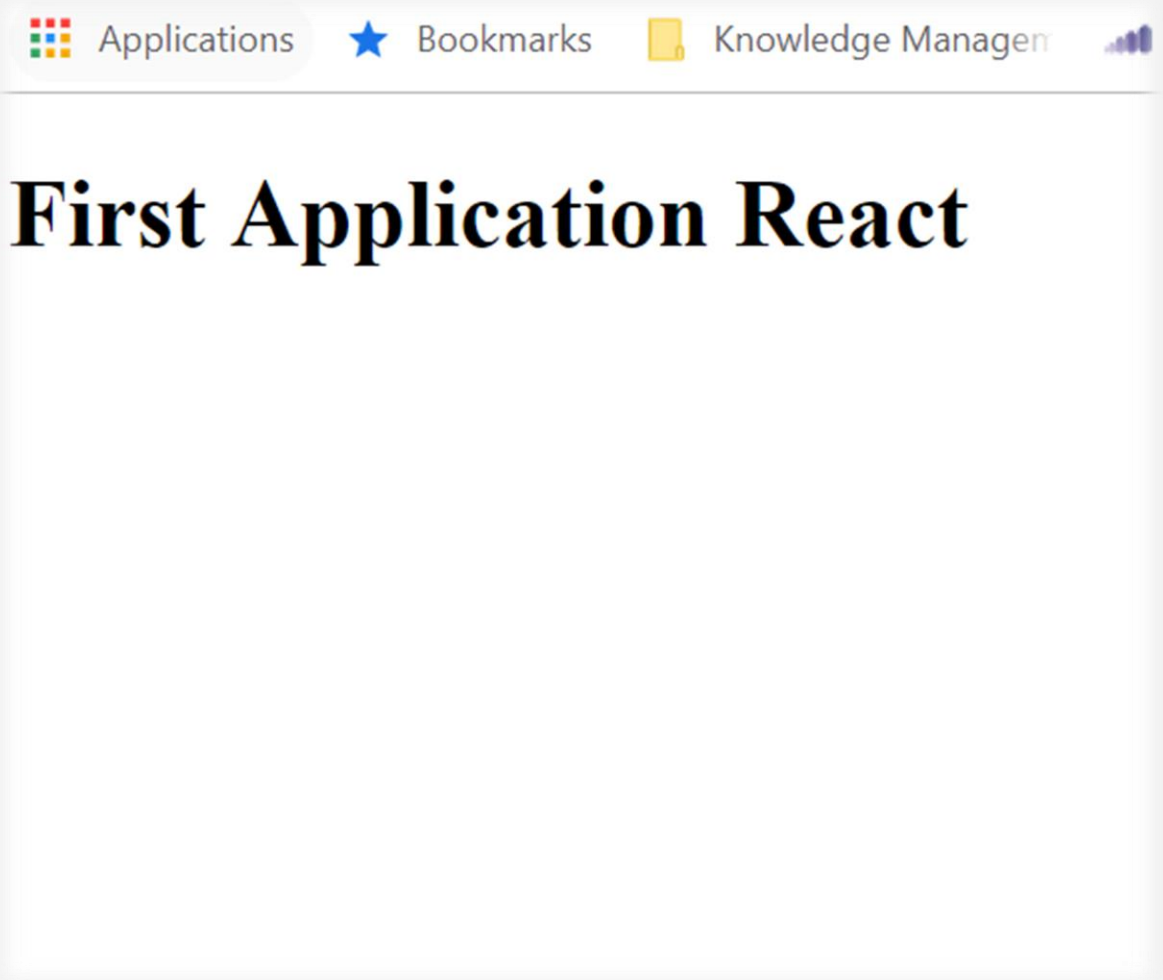


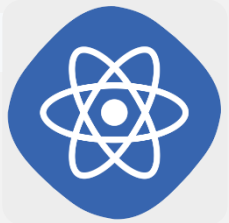
A vous de jouer !

{} Etape 7 : Créer un composant **Contact.js**, pour lui afficher, son nom, prénom, numéro de téléphone, et email.

- Le composant doit avoir **un état d'initialisation contenant toute ces propriétés**. Ce dernier doit être ajouter au composant **App.**

A vous de jouer !





ReactJS

Vers un monde meilleur !

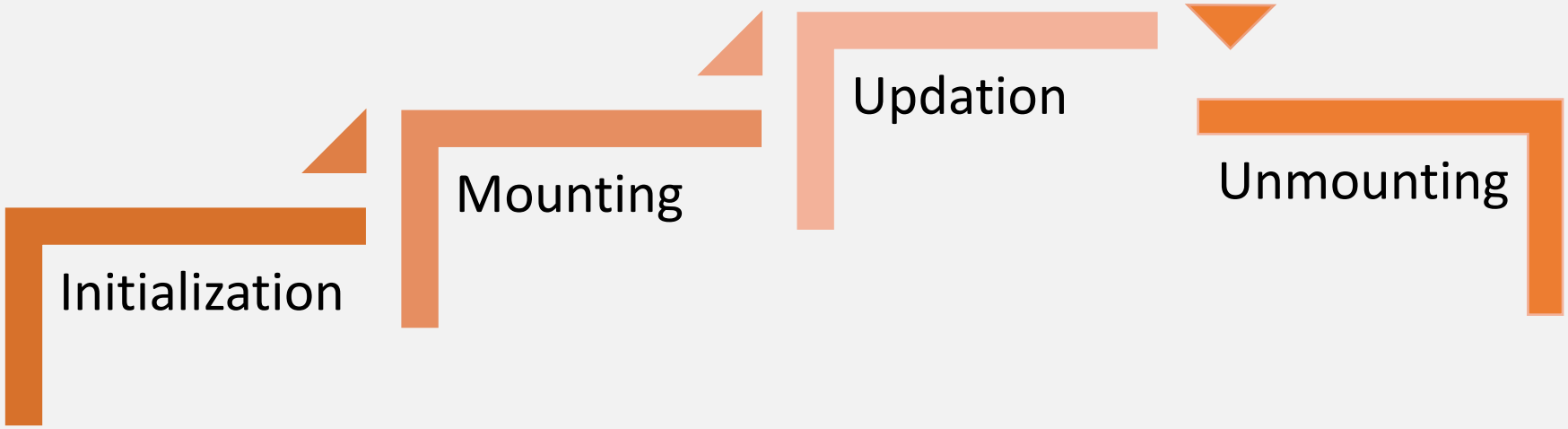
Gestion du cycle de vie avec **les hooks**

ReactJS

Vers un monde meilleur !

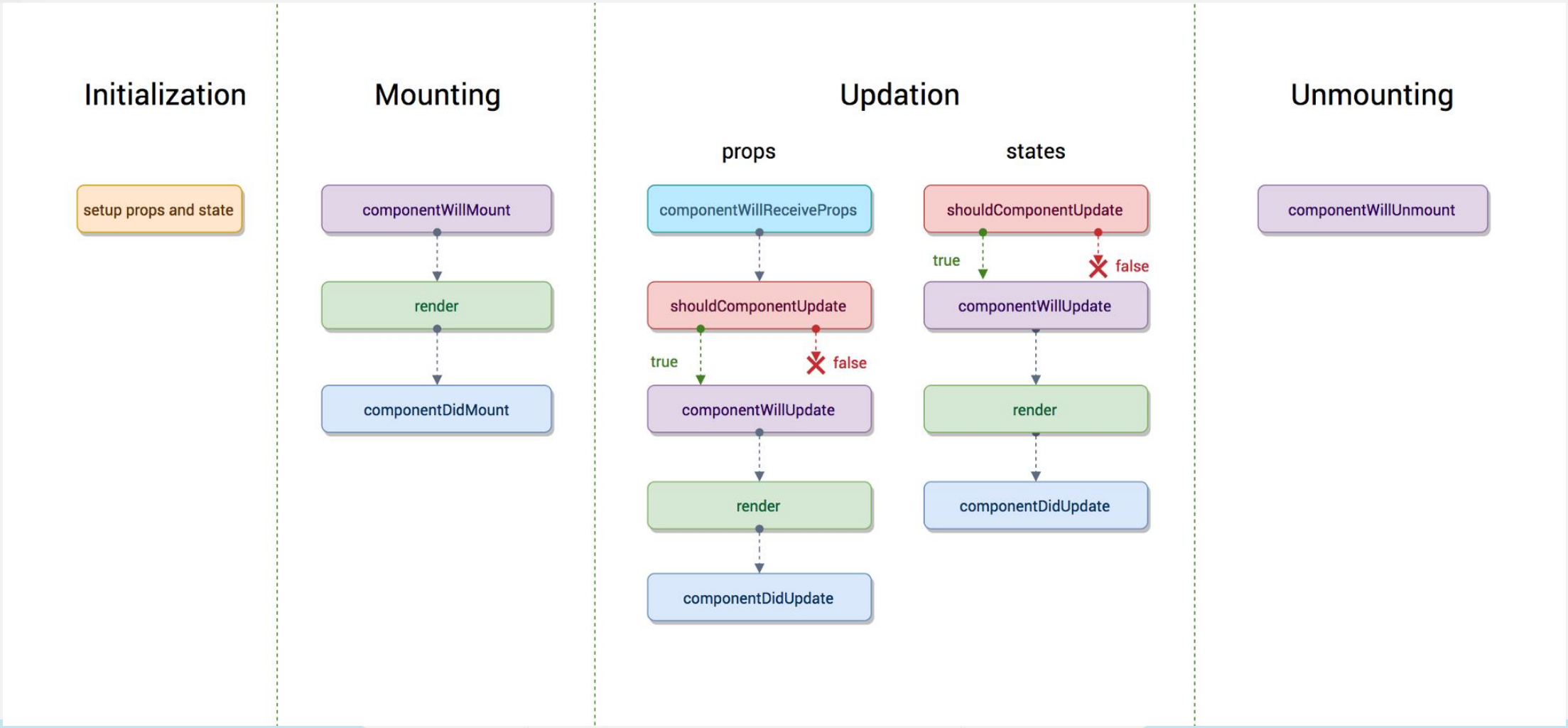
Gestion du cycle de vie des états avec les *Hooks*

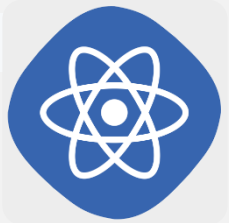
- 4 phases d'un composant React



ReactJS

Vers un monde meilleur !





ReactJS

Vers un monde meilleur !

- **Setup** props and state

```
class Contact extends Component {
   constructor(){
      super();
      this.state = {
      firstName: '',
      lastName: ''
   };
Contact.defaultProps = { theme:
'dark' }
}}
```

- *Setup du state initiale dans le **constructeur***
- *Setup des props par défaut par la propriété **defaultProps***

ReactJS

Vers un monde meilleur !

- **Mounting**

  - ❑ componentWillMount
    - Exécuté juste avant la monté du composant dans le DOM
    - Exécuté juste avant le première appel de la méthode **Render()**

  - ❑ render
    - Méthode pure, pour la monté du composant dans le navigateur.

ReactJS

Vers un monde meilleur !

- **Mounting**

  - ❑ componentDidMount

    - Exécuté juste après la monté du composant dans le DOM
    - Utile pour la récupération des données par appel d'API *(par exemple)*

ReactJS

Vers un monde meilleur !

- **Updation**
  - Cette étape survient lorsque le composant reçoit une mise à jour
  - Démarrage de cette étape lors de l'appel de la méthode setState()

❑ souldComponentUpdate

  - Dit à React que le composant **reçoit une nouvelle demande de mise à jour** des props **et ou** state.
  - Cette dernière porte bien son nom : Est-ce que le composant doit être mise à jour.

ReactJS

Vers un monde meilleur !

- **Updation**

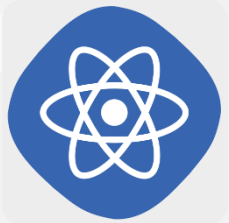
  - ❑ componentWillUpdate

    - Exécute **après** que la méthode **shouldComponentUpdate** renvoie **true**
    - Appel de la méthode render, une fois cette dernière exécutée.

  - ❑ componentDidUpdate

    - Est exécuté quand la mise à jour a été effectué dans le DOM

    - Cette méthode est utilisée pour relancer les bibliothèques tierces utilisées pour s'assurer que ces bibliothèques se mettent également à jour et se rechargent.



ReactJS

Vers un monde meilleur !

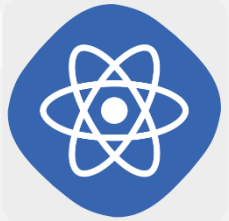
- **Updation**
  - Liste des méthodes qui seront appelées lorsque le parent enverra de nouvelles props sont les suivantes :
    - ❑ componentWillReceiveProps
      - Est exécuté lorsque les props ont changées et ne sont pas d'abord rendus.

ReactJS

Vers un monde meilleur !

- **Unmounting**
  - Dans cette phase, le composant n'est pas nécessaire et il sera « démonté » du DOM. La méthode appelée dans cette phase est la suivante :
    - ❑ componentWillUnmount
      - Dernière méthode du cycle de vie
      - Exécuté avant que le composant soit démonté du DOM
      - Utile pour nettoyer par exemple des informations sensibles des informations d'authentification

A vous de jouer !

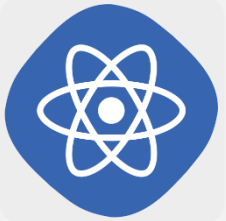


A vous de jouer !

{} Etape 1 : Créer un dossier **Step1** dans le dossier **src**, puis y placer les fichiers **main, App, et Contact.js**

{} Etape 2 : Créer un dossier **Step2** et y placer la copie des trois fichiers précédemment déplacés

{} Etape 3 : Ajouter un nouveau composant **AllContacts.js**. Dans celui-ci placer un **hook** permettant la récupération de tous les contacts lorsque le composant est chargé. **(Utiliser le serveur node du TP sur l'ES6).**

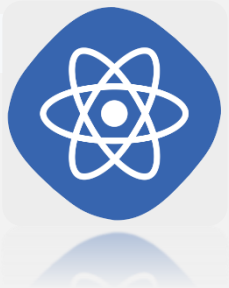


A vous de jouer !

{} Pour informations, faire un git pull de la dernière version du git sur EcmaScript6 experience sur la branch Feat/finalStep

{} Etape 4 : Dans le composant, AllContact, afficher pour chaque contact ses informations **(réutilisation du composant contact obligatoire)**

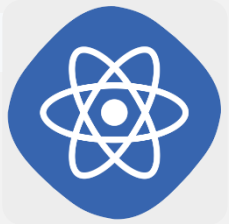
{} Etape 5 : **Modification de l'entrée** dans le fichier **webpack.common.js** pour la faire **pointer** sur notre dossier **step2**



ReactJS

Vers un monde meilleur !

PropTypes

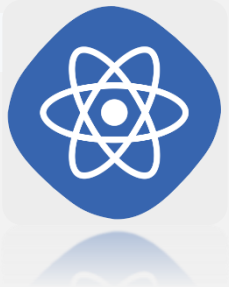


ReactJS

Vers un monde meilleur !

Définir fortement les props avec PropTypes

- Probablement aucune erreur, aucun avertissement dans la console
- Pour configurer correctement le composant, il faut lui passer les bonnes *props*.
- Les *props* sont, véritablement, **l'API du composant**. Pour cela, il est fondamental de définir formellement cette API (la liste des props).



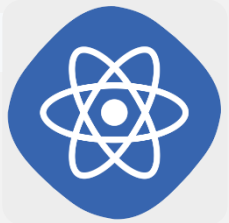
ReactJS

Vers un monde meilleur !

Définir fortement les props avec PropTypes

❑ Comment ça fonctionne ???

- React examine sur chaque composant une propriété statique nommé **propTypes**.
- C'est un objet **dont les clés sont les noms des props attendues**, et les valeurs des validateurs de props.
- Le module standard **prop-types** fournit une série de validateur.



ReactJS

Vers un monde meilleur !

Définir fortement les props avec PropTypes

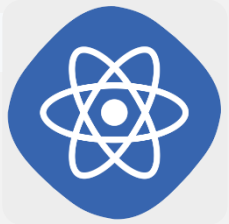
- Le développeur pourra avoir un retour instantané si une props est requise ou si la valeur n'est pas du bon type par exemple.

❑ Mise en place

```
import Proptypes from 'prop-types';
```

Juste avant l'export du composant

```
Contact.propTypes = {
.....
};
```



ReactJS

Vers un monde meilleur !

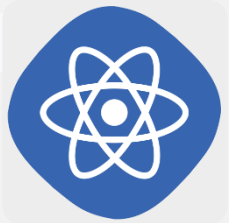
Définir fortement les props avec PropTypes

- Sans mécanisme de définition formel, les bugs sont parfois difficile à repérer

```
<Contact props={{firstName : 'toto'}} />
```

❑ **Exemple :**

```
<Contact Props={{firstName : 'toto'}} />
```

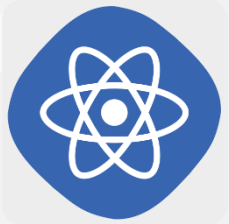


A vous de jouer !

{} Etape 1 : Créer un dossier **Step3** dans le dossier **src**, puis y placer les fichiers **main, App, et Contact.js précédemment modifié dans l'étape 2**

{} Etape 2 : **Enrichir le composant Contact** afin que si une propriété n'est pas remplie tel que firstName, une erreur apparaîtra dans le navigateur.

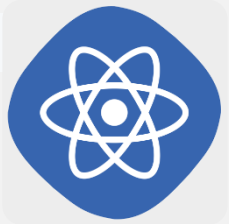
{} Etape 3 : **Modification de l'entrée** dans le fichier **webpack.common.js** pour la faire **pointer** sur notre dossier **step3**



ReactJS

Vers un monde meilleur !

Orienté **Composant**
StateLess



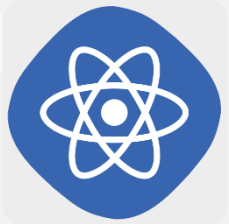
ReactJS

Vers un monde meilleur !

Orienté composant : *StateLess*

```
import React from 'react';

const Contact = (props)=>{
   return (
      <div>
      je suis un composant
      Contact {props.firstName}
      </div>
   );
}
```

- Fonction ayant la signature **`object => JSX`**
- Composant **sans gestion de son propre état.**
- **Props est passé par le parent.** Ils sont qu'en lecture seule.



ReactJS

Vers un monde meilleur !

Orienté composant : *StateLess*

- Pour passer une valeur à un composant :

```
<Contact props={{firstName : 'toto'}} />
```

- Le composant créé est aussi appelé « Composant fonctionnel » car il est créé en utilisant une fonction pure.

- Le résultat ne dépend que des arguments reçus et rien d'autre

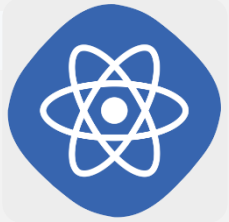
ReactJS

Vers un monde meilleur !

HOC : Higher Order Component

- Kesako ???

  - C'est une fonction de base qui renvoie une autre fonction améliorée.



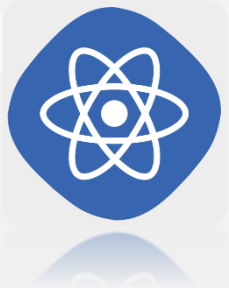
ReactJS

Vers un monde meilleur !

HOC : Higher Order Component

❑ Le but ??

- Rendre le code plus lisible en cachant la logique derrière une fonction
- Appelé aussi « pattern décorateur »



ReactJS

Vers un monde meilleur !

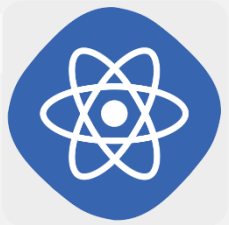
HOC : Higher Order Component

❑ **Exemple :**

- Affiche un spinner de chargement tant que le contact n'est pas chargé

```
const Contact = (props) =>
 {
   return props.loading
   ? <StylishSpinner />
   : <div>
   {props.firstName}
   </div>
}
```

- **Problème** : L'affichage qui apporte de la valeur ***(comment afficher un utilisateur)*** est noyée avec la partie qui s'occupe de l'affichage du chargement.



ReactJS

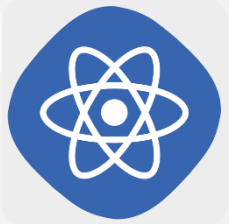
Vers un monde meilleur !

HOC : Higher Order Component

Première extraction

```
const FillContact = (props) =>
{
   return (
      <div>
      {props.user.name}
      </div>
   )
}
```

```
const Contact = (props) => {
return props.loading
? <loadSpinner />
: <FillContact {...props} />
}
```



ReactJS

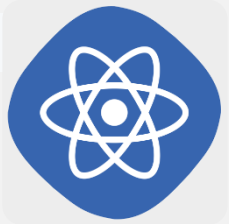
Vers un monde meilleur !

HOC : Higher Order Component

Deuxième extraction

```
const withLoading =(isLoading,
BaseComponent)=> {
   return (props)=> {
      isLoading(props)
      ? <StylishSpinner />
      : <BaseComponent {...props} />
   }
}
```

```
const Contact =
withLoading(
      (props) =>
      props.loading,
      FillContact
)
```



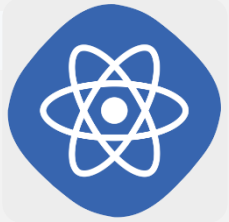
A vous de jouer !

{} Etape 1 : Créer un dossier **Step4** dans le dossier **src**, puis y placer les fichiers **main, App, et Contact.js précédemment modifié dans l'étape 3**

{} Etape 2 : Ajout d'un fichier **ContactHoc.js** dans le dossier **Step4**

{} Etape 3 : Transformer le composant Contact en composant stateless

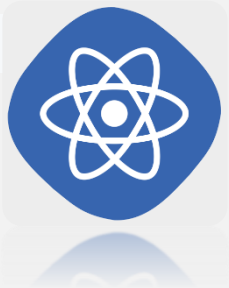
. *Pour information ce composant ne disposera pas d'état propre.*



A vous de jouer !

{} Etape 4 : Modification du fichier **App.js** pour ajouter notre nouveau composant **stateLess**

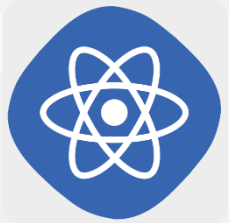
{} Etape 5 : **Modification de l'entrée** dans le fichier **webpack.common.js** pour la faire **pointer** sur notre dossier **step4**



ReactJS

Vers un monde meilleur !

La bibliothèque **Recompose**

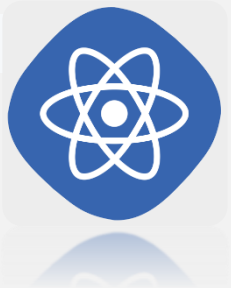


ReactJS

Vers un monde meilleur !

Recompose :

- ❑ Kesako ???
  - Assistant de gestion d'état pour les composants HOC
  - Dispose de nombreuse méthode d'aide pour la gestion des états

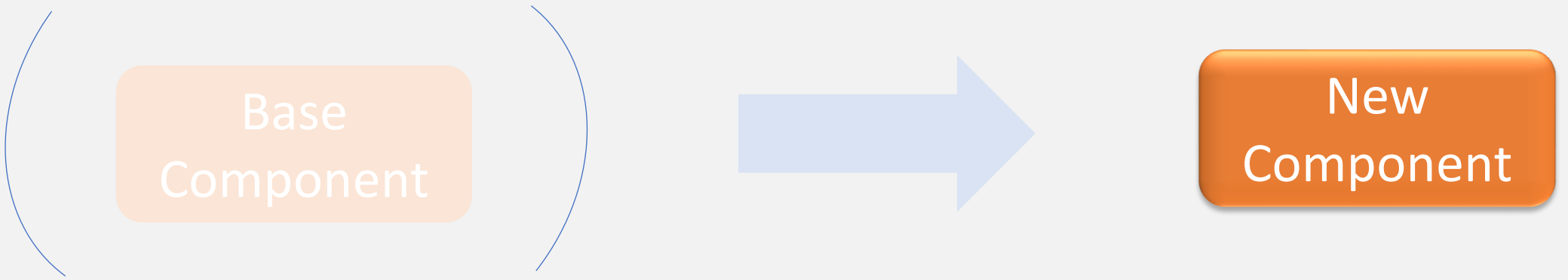


ReactJS

Vers un monde meilleur !

Exemple de HOC les containers :

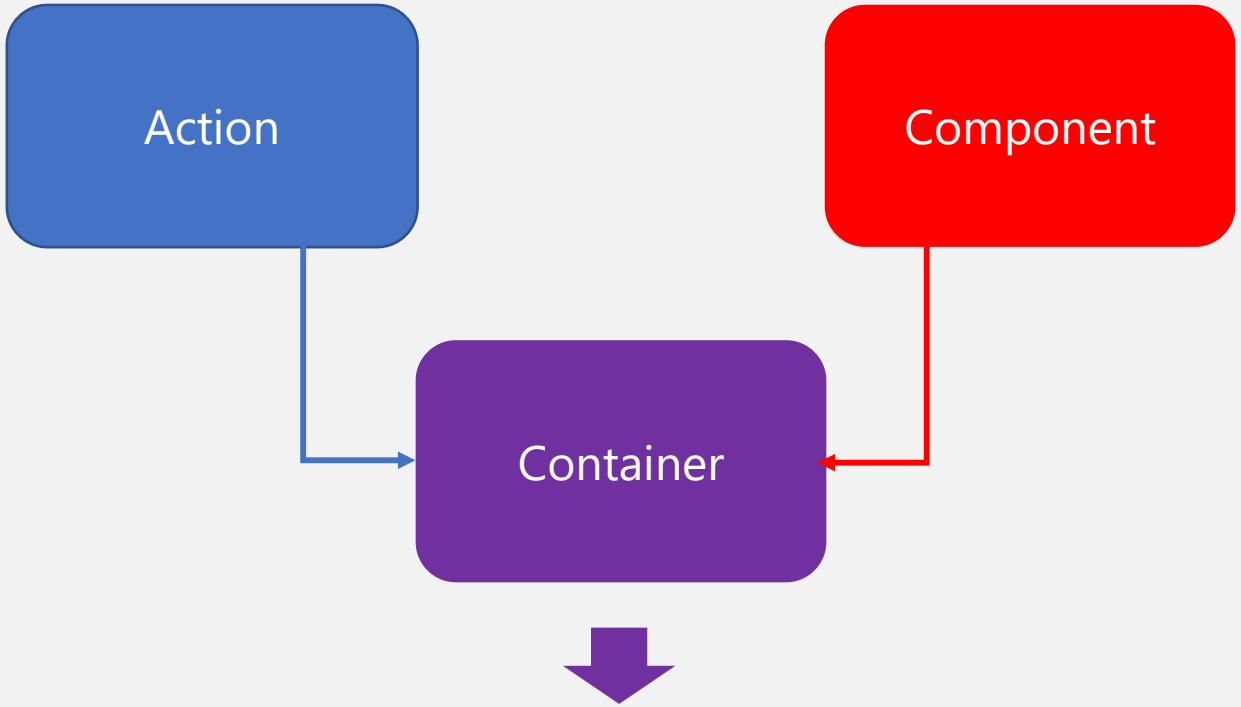
- Kesako ???

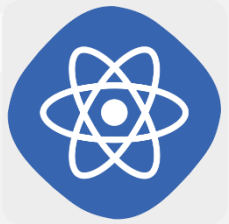


ReactJS

Vers un monde meilleur !

Exemple de HOC les containers :



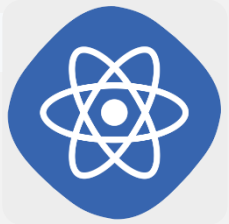


ReactJS

Vers un monde meilleur !

Recompose :

- compose()
  - Composition de plusieurs HOC pour les fusionner en un seul
- withHandlers()
  - Ajoute de nouveau handler d'événement comme propriété du composant encapsulés

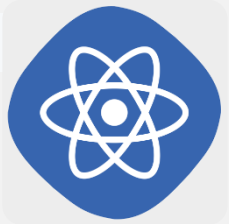


ReactJS

Vers un monde meilleur !

Recompose :

- withState()
  - Ajout d'un état local au composant
- Elle prend en paramètre :
  - Le **nom de l'attribut dans l'état** qui sera donné comme propriété au composant enfant
  - Le **nom de la propriété contenant la fonction** pour mettre à jour cette état
  - La valeur initiale (**statique ou fonction** prenant en paramètre les propriétés et retournant la valeur initiale)



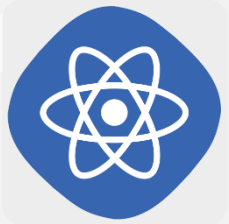
ReactJS

Vers un monde meilleur !

Recompose :

- withState()

```
withState(
   "inputValue",
   "setInputValue",
   // `formatValue` est l'un des paramètres de
   notre HOC
   props => formatValue(props.value),
);
```



ReactJS

Vers un monde meilleur !

Recompose :

- Tout comme le composant StateFull, il est possible de gérer le cycle de vie d'un composant

❑ lifeCycle()

- Ajout d'un « hook » permettant d'utiliser une partie du cycle de vie d'un composant.

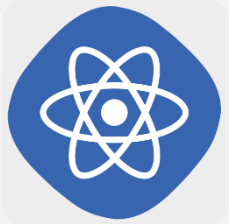
```
lifecycle({
        componentDidMount: ()=> {
        this.props.getList(this.props);
} }
```

A vous de jouer !

{} Etape 1 : Créer un dossier **Step5** dans le dossier **src**, puis y placer les fichiers **main, App, et ContactHoc.js précédemment modifiés dans l'étape 4**

{} Etape 2 : Création d'un fichier **Contact.container.js**. Dans ce dernier importer le module **recompose**.

{} Etape 3 : Modifier le composant HOC en vue d'ajouter une zone de saisie pour changer le nom de du contact



A vous de jouer !

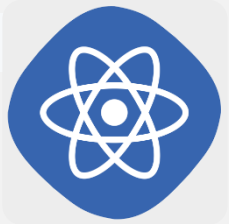
{} Etape 4 : Enrichir le composant HOC via le container créé précédemment. Ajouter une méthode permettant de changer le nom de l'utilisateur.

> **Astuce : Utiliser la méthode compose, withState, withHandler, mapProps**

{} Etape 5 : Transformer le composant AllContact de l'étape 2 en HOC.

Création d'un nouveau fichier container permettant d'enrichir ce composant.

Placer la méthode de gestion du cycle de vie **componentDidMount** dans ce container.

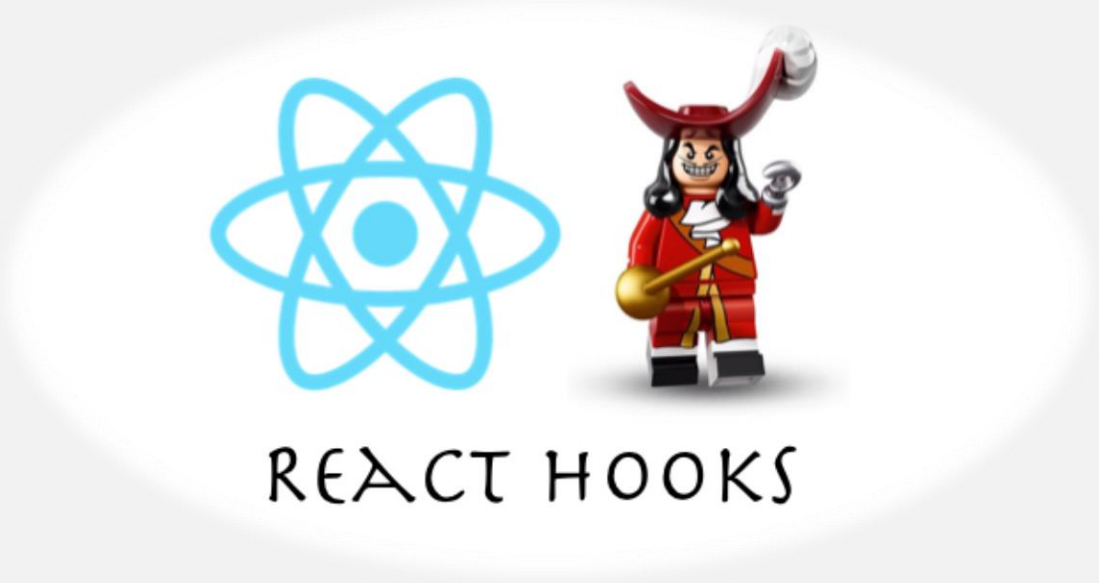


A vous de jouer !

{} Etape 5 : **Modification de l’entrée** dans le fichier **webpack.common.js** pour la faire **pointer** sur notre dossier **step5**

ReactJS

Vers un monde meilleur !



ReactJS

Vers un monde meilleur !

Hooks API : *le futur de React*

❑ Kesako ???

- Disponible depuis la **version 16.8** de React.JS
- Permet :
  - D'écrire du code plus propre et épuré.
  - De ce passer complétement du mot clé class
  - D'utiliser la gestion d'état centralisé dans des composants "Stateless".

ReactJS

Vers un monde meilleur !

useState

useEffect

Live Coding

useContext

useEffect

ReactJS

Vers un monde meilleur !

Hooks API: *le futur de React*

- **useState** :
  - Permet d'ajouter un état à un composant
- **useEffect** :
  - Permet d'ajouter un évenement de gestion du cycle de vie (componentDidMount, componononentDidUpdate).
- **useReducer** :
  - Permet d'ajouter un reducer permettant de regrouper le traitement des données
- **useContext** :
  - Permet d'ajouter un context global à l'application évite l'effet « props drilling »

A vous de jouer !

- Mettre en place une gestion d’état centralisé.
- Avec la méthode useEffet, récupérer les contacts du serveur de manière asynchrone afin de les ajouter dans l’état centralisé
- Transformer les composants afin de récupérer les données de l’état centralisé