

React.JS

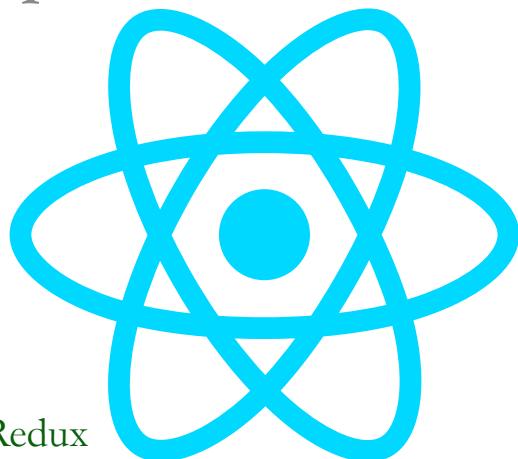
Formation Stéphane Brunet

www.chapoulougne.com



React.JS

un framework JavaScript



- ❑ Rappels des composants des RIA
 - ❑ Développer avec ReactJS
 - ❑ Interactivité des composants
 - ❑ Application monopage avec ReactJS Redux
 - ❑ Application isomorphique
 - ❑ Introduction à React Native

développer patterns propriétés formulaire différences gestionnaire
dispatcher initiation ecosystème pattern
loader logique bénéfices comparaison architectures suivre
logique extension données mise œuvre transfert stores états routes
œuvre réactifs dom data html natifs
vie redux spa hot ide ins création vues css ensemble
api ria jsx js ide mvc os actions vues css ensemble
virtual react rôle mvc ui création vues css ensemble
plug plug état état cycle cycle
état état cycle cycle
comprendre monopage détails native
design composant contrôleur design
composant contrôleur design
native flow détails native
flow détails native
native flow détails native
selon réduire selon réduire
serveur définition cordova environnement selon approche
transpilers node principe limitations fonctionnel

React.JS

un framework JavaScript

- Rappels des composants des RIA
- Développer avec ReactJS
- Interactivité des composants
- Application monopage avec ReactJS Redux
- Application isomorphique
- Introduction à React Native



- Les fondamentaux. HTML, CSS, JavaScript. Le DOM.
- IHM du Javascript
- Ecosystème des frameworks JavaScript.
- Principes de Data-Binding : dirty-checking, observable, virtual-dom..
- Workers

React.JS

un framework JavaScript

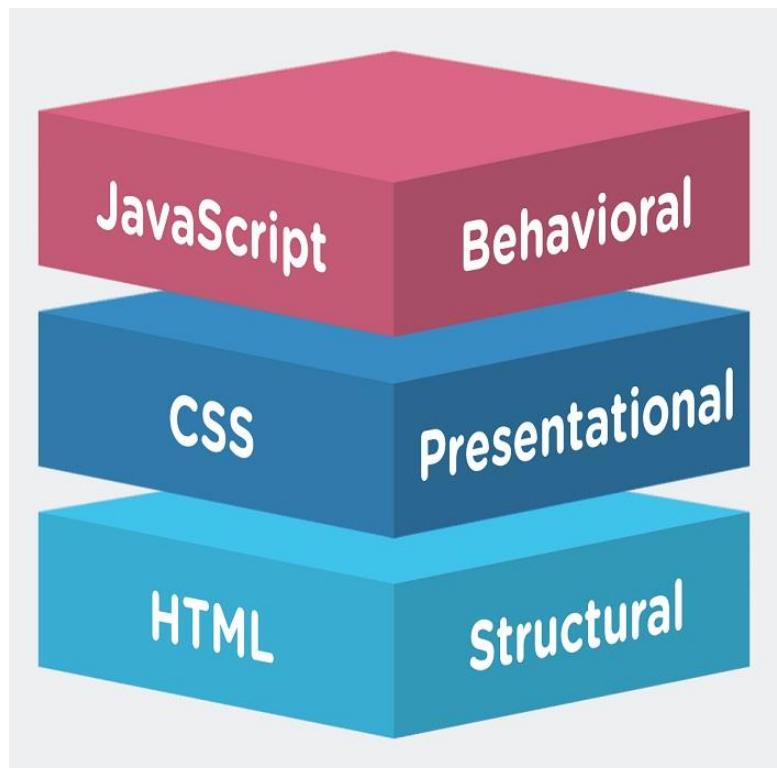
Introduction

Présentation

Javascript



Depuis sa création, le JavaScript n'a cessé de croître dans l'usage. Plus qu'un « langage de scripts basiques », il est devenu ces dernières années un langage absolument incontournable pour toute personne s'intéressant à la création de sites web. Il est désormais entré dans la cour des grands, aux côtés d'HTML, de CSS 3



React.JS

un framework JavaScript

Introduction

Présentation

Javascript



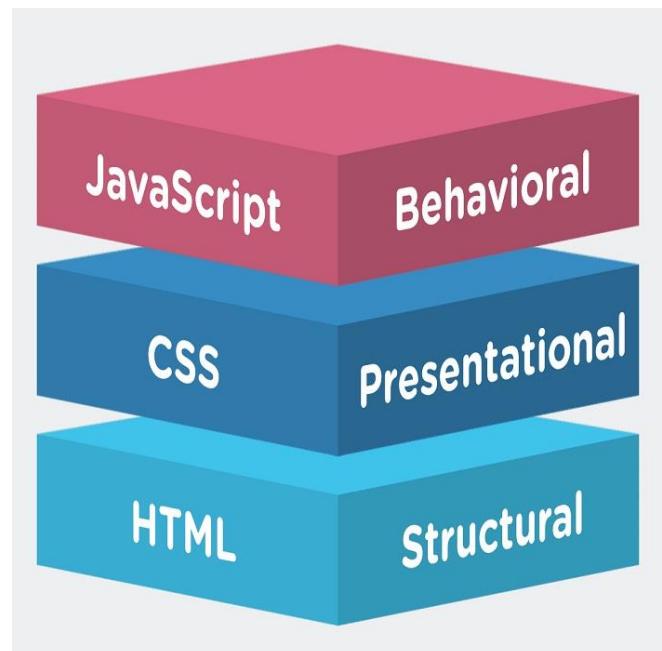
Le JavaScript est majoritairement utilisé sur Internet, conjointement avec les pages web

HTML dans lesquelles il est inclus (ou dans un fichier externe).

Le JavaScript permet de « dynamiser » une page HTML en ajoutant des interactions avec l'utilisateur, des animations, de l'aide à la navigation.

Par exemple :

- afficher/masquer du texte ;
- faire défiler des images ;
- créer un diaporama avec un aperçu « en grand » des images;
- créer des infobulles.



React.JS

un framework JavaScript

Introduction

Présentation

Javascript



JavaScript c'est un langage interprété: dans ce cas, il n'y a pas de compilation. Le code source reste tel quel, et si l'on veut l'exécuter, on doit le fournir à un interpréteur qui se chargera de le lire et de réaliser les actions demandées.

Chaque navigateur possède un interpréteur JavaScript propre :

- pour Internet Explorer, il s'agit de Chakra (l'interpréteur JavaScript des versions antérieures à IE 9 est JScript) ;
- pour Mozilla Firefox, l'interpréteur se nomme SpiderMonkey;
- pour Google Chrome, il s'agit de V8.



React.JS

un framework JavaScript



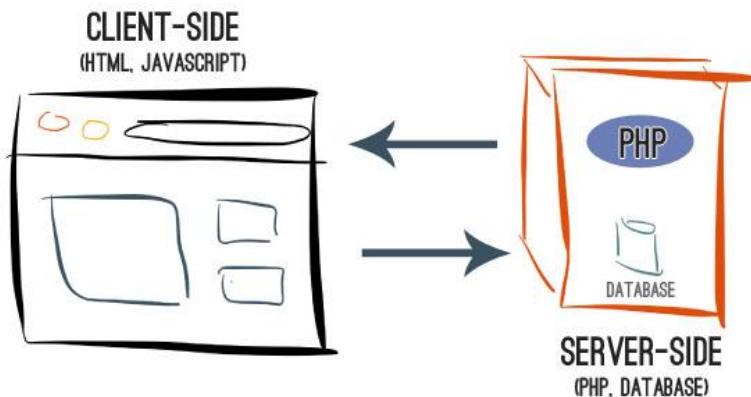
Introduction

Présentation

Javascript

Le JavaScript est un langage dit client-side, c'est-à-dire que les scripts sont exécutés par le navigateur de l'internaute (le client).

Cela diffère des langages de scripts dits server-side qui sont exécutés par le serveur web. C'est le cas des langages comme le PHP ou .net



React.JS

un framework JavaScript

Introduction

Présentation

Javascript



Brendan Eich développe le Live Script, un langage de scripts qui s'inspire du langage Java, et qui est destiné à être installé sur les serveurs développés par Netscape.

Netscape se met à développer une version client du Live Script, qui sera renommée JavaScript en hommage au langage Java créé par la société Sun Microsystems. En effet, à cette époque, le langage Java était de plus en plus populaire, et appeler le LiveScript « JavaScript » était une manière de faire de la publicité à la fois au Java et au JavaScript lui-même.

React.JS

un framework JavaScript

Introduction

Présentation

Javascript



L'ECMAScript et ses dérivés

L'ECMAScript est la référence de base dont découlent des implémentations. On peut citer le JavaScript, qui est implémenté dans la plupart des navigateurs que est le script qui supporté par tous les navigateurs

Les versions du JavaScript sont basées sur celles de l'ECMAScript

- ES 1 et ES 2: sont les prémisses du langage JavaScript;
- ES 3 (sorti en décembre 1999) ;
- ES 4, qui a été abandonné en raison de modifications trop importantes ;
- ES 5 (sorti en décembre 2009);
- ES 6 finalisé en décembre 2014

Ko Koa	Sb Storybook	D DB	Jq jQuery	We Webs	Rn React Native	Ko Koa	Me Meteor	Nx Next.js	G Gatsby	Av Avast	Ez Enzyme	Sb Storybook	D DB	Jq jQuery	We Webs	Rn React Native	Ko Koa	Sb Storybook
20 Pr React	29 Bb Babel	20 Pc Parcel	30 Gu Gulp	35 Io Ionic	18 Fw Flow	20 Pr React	20 Em Ember	27 Bb Backbone	112 V Vue.js	40 Ng Angular	44 Rd Redux	29 Bb Babel	20 Pc Parcel	30 Gu Gulp	35 Io Ionic	18 Fw Flow	20 Pr React	29 Bb Babel
38 Ts TypeScript	11 Ry Relay	44 Wp Webpack	38 Mt Moment	15 Ns Node.js	7 Cs ClosureScript	38 Ts TypeScript	1 St Stencil	2 At Angular	3 E Element	10 R React	4 Of Object	1 Ry Relay	44 Wp Webpack	38 Mt Moment	15 Ns Node.js	7 Cs ClosureScript	38 Ts TypeScript	11 Ry Relay
40 Ex Expo	12 Lb Laravel	8 Ap Angular	34 Lo Loopback	64 E Ember	6 Re React	40 Ex Exponent	5 Ja Jade	6 Va Vanilla	7 Sc Svelte	8 Ri React	9 Pt Preact	12 Lb Laravel	8 Ap Angular	34 Lo Loopback	64 E Ember	8 Re React	40 Ex Exponent	12 Lb Laravel
20 Sa Sails	10 Ka Knockout	14 Ra React	24 Un Unidirectional	2 Cv Cypress	4 Em Ember	20 Sa Sails	20 Je Jeopardy	16 Mo MongoDB	10 18 18	11 Av Avatar	14 Ez Enzyme	10 Ja JavaScript	14 Ka Knockout	24 Un Unidirectional	2 Cv Cypress	4 Em Ember	20 Sa Sails	10 Ka Knockout
22 Ko Koa	28 Sh Shaka	70 D Django	60 Un Unidirectional	10 Cv Cypress	28 Re React	28 Ro Rails	40 Me Meteor	25 Nx Next.js	25 G Gatsby	16 Av Avatar	16 Ez Enzyme	20 Si Simplon	25 D Django	20 Un Unidirectional	20 Cv Cypress	20 Em Ember	22 Ko Koa	28 Sh Shaka

React.JS

un framework JavaScript



Introduction

Présentation

Javascript

Javascript est utiliser dans le code d'applications web critiques et les technologies d'assistance doivent pouvoir interpréter le contenu dynamique

- Toute fonctionnalité JavaScript doit prendre une forme qui peut être interprétée comme du texte.
- Toute fonctionnalité JavaScript doit être accessible au clavier.
- Les activités temporelles en JavaScript doivent pouvoir être contrôlées par l'utilisateur, à moins que cela ne change leur sens

Ko Ko	Sb Storybook	D DS	Jq Quality	We Wewi	Rn React Native	Ko Ko	Me Meteor	Nx Next.js	G Gatsby	Av Avia	Ez Enzyme	Sb Storybook	D DS	Jq JQuery	We Wewi	Rn React Native	Ko Ko	Sb Storybook	
Pr Preact	Bb Babel	Pc Parcel	Gu Gulp	Io Ionic	Fw Flow	Pr Preact	Em Ember	Bb Backbone	V Vue.js	Ng Angular	Rd Redux	Bb Babel	Pc Parcel	Gu Gulp	Io Ionic	Fw Flow	Pr Preact	Bb Babel	
Ts TypeScript	Ry Relay	Wp Webpack	Mt Moment	Ns Nativescript	Cs ClojureScript	Ts TypeScript	St	At At	E E	Mo Mo	R React	Of Of	W W	Ry Relay	Wp Webpack	Mt Moment	Ns Nativescript	Cs ClojureScript	Ts TypeScript
Ex Express	Lb Loopback	Ap Apollo	Lo Lodash	E Express	Re React	Ex Express	Ja Ja	Va Va	Sc Sc	Ri Ri	Pt Pt	Lb Loopback	Ap Apollo	Lo Lodash	E Express	Re React	Ex Express	Lb Loopback	
Sa Sails	Ka Koa	Ra Ramda	Un Underscore	Cv Circular	Em Elm	Sa Sails	Je Jest	Mo Mo	20 20	18 18	Ja Ja	Ka Koa	Ra Ramda	Un Underscore	Cv Circular	Em Elm	Sa Sails	Ka Koa	
Kn Knex	Sh Shaka	P Preact	Tr Trino	Re Relay	Ro Ro	Me Meteor	Nx Next.js	G Gatsby	Av Avia	Ez Enzyme	Sj Svelte	D D	Jq JQuery	We Wewi	Rn React Native	Ko Ko	Sb Storybook		

React.JS

un framework JavaScript

Introduction

Présentation

Javascript



RESSOURCE Éditeurs web

SublimeText (Multiplateforme, payant)

<http://www.sublimetext.com/>

Brackets

Atom

WebStorm

Visual studio Code

RESSOURCE Navigateurs web

Une console de débogage est sur tous les navigateurs modernes **F12**

Avec un ensemble d'outils pour "comprendre" le code interprété par le navigateur.

React.JS

un framework JavaScript

Introduction

IHM

Javascript



Structure des instructions

Un langage proche du C

- Chaque instruction se termine par un point-virgule
- On peut avoir plusieurs instructions par ligne
- On peut avoir une instruction sur plusieurs lignes
- Un instruction est soit un appel à une fonction soit une opération avec un opérateur
- Les commentaires sur une ligne commencent par // ou <!--
- Les commentaires sur plusieurs lignes sont encadrés par /* et */

React.JS

un framework JavaScript

Introduction

IHM

Javascript

Intégration dans une page HTML



Un élément <script> est présent : c'est lui qui contient le code JavaScript

Il s'agit d'une instruction, c'est-à-dire une commande, une action que l'ordinateur va devoir réaliser.

Les langages de programmation sont constitués d'une suite d'instructions qui, mises bout à bout, permettent d'obtenir un programme ou un script complet.

```
<html>
  <head>
    <title>Test JavaScript</title>
  </head>
  <body>
    Texte de la page
    <script>
      var date= new Date();
      var texte= "Nous sommes le: "+date;
      document.write( texte);
    </script>
  </body>
</html>
```



React.JS

un framework JavaScript

Introduction

IHM

Javascript

Intégration dans une page HTML



L'élément <script> peut aussi appeler un fichier externe

Il s'agit d'une instruction, c'est-à-dire une commande, une action que l'ordinateur va devoir réaliser.

Les langages de programmation sont constitués d'une suite d'instructions qui, mises bout à bout, permettent d'obtenir un programme ou un script complet.

```
<html>
  <head>
    <title>Test JavaScript</title>
  </head>
  <body>
    Texte de la page
    <script src="MonFichier.js">
  </body>
</html>
```

React.JS

un framework JavaScript

Introduction

IHM

Javascript

Intégration dans une page HTML



Il est aussi possible d'appeler des fonctions directement dans mon code html

Cette méthode sera utiliser pour des fonctions très simples et on préfère séparer le html du js

```
<a href="javascript:nomFonction()">Cliquer ici</a>
```



React.JS

un framework JavaScript

Introduction

IHM

Javascript

Intégration dans une page HTML ou pas



Si un navigateur n'est pas capable d'exécuter le code JavaScript, ou le javascript est bloqué par default (1% des internautes) prévoir un message entre les balises <NOSCRIPT>.

```
<NOSCRIPT>
```

Attention, cette page utilise normalement le langage JavaScript qui n'existe pas ou est désactivé sur votre navigateur.


```
</NOSCRIPT>
```

React.JS

un framework JavaScript

Introduction

IHM

Javascript



Trois possibilités, plus ou moins pauvres

Avant en javascript, lorsqu'on avait besoin de contrôler la valeur d'une variable on faisait un "alert(mavariable)".

Maintenant il existe une méthode bien plus élégante qui consiste à utiliser la log du navigateur

console.debug() remplacé par console.log()

console.dir()

console.error()

console.group()

console.groupCollapsed()

console.groupEnd()

console.info()

console.time()

console.timeEnd()

console.trace()

console.warn()

```
console.log(mavariable);
console.time("Test Timer");
console.dir();
console.group("test groupe");
console.log("une indentation");
console.groupCollapsed("avec un possibilité de réduit");
console.groupEnd();
console.info(fruit);
console.warn("test warning");
console.error("Test Message Erreur");
console.timeEnd("Test Timer");
```

React.JS

un framework JavaScript

Introduction

IHM

Javascript



ES6 a été pensé pour créer des applications web facilement maintenables, tout en restant compatibles avec le code existant. L'idée a été d'ajouter de nouvelles fonctionnalités au langage.

Ainsi, la bibliothèque standard s'enrichit de nouvelles méthodes, mais surtout, le langage adopte de nouvelles syntaxes, comme les classes ou les modules (pour ne citer qu'eux) permettant d'avoir du code beaucoup plus structuré et lisible



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Déclaration d'une constante avec le mot **const**

```
const nom = "Brunet";
console.info (nom);
nom = "Bond" ;// erreur
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Le mot clé **let** permet de déclarer une variable limitée à la portée d'un bloc, c'est-à-dire qu'elle ne peut être utilisée que dans le bloc où elle a été déclarée, ce qui n'est pas le cas avec var.

```
function swap(x, y) {  
  if (x != y) {  
    var old = x;  
    let tmp = x;  
    x = y;  
    y = tmp;  
  }  
  console.info(typeof(old));  
  console.info(typeof(tmp));  
}  
swap(2,3);
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Le **Template String** va nous permettre de déclarer des chaînes avec des variables à évaluer à l'intérieur. À la différence d'une chaîne de caractères classique, on utilise des back quotes (`) à la place des double quotes.

```
let me = "James Bond";
let macredit = 7;
let mi6 = `Je suis ${me} agent 00${macredit}`;
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Lors de l'écriture d'une fonction qui doit additionner deux nombres.
Si les paramètres ne sont pas définis alors votre programme plante,
pour éviter cela

```
let add = function (x = 0, y = 0) {  
    return x + y;  
};
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



L'utilisation d'une fonction asynchrone modifie le comportement de notre fonction .

Avec ES 6

```
var obj={  
  noms :["M", "Q", "Bond"],  
  log : function(){  
    setTimeout(() =>{  
      console.log(this.noms);  
    }, 10 );  
  }  
}
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Il n'existe toujours pas de mot clé `foreach` en JavaScript 2015, cependant il y a désormais une nouvelle façon d'itérer sur un tableau en utilisant le couple **for-of**.

```
let languages = [ "C", "C++", "C#", "JavaScript" ];
for (let language of languages) {
    console.log(language);
}
```

Il est aussi possible d'utiliser un système clé/valeur en spécifiant un tableau en paramètre et en exploitant la fonction `Array::entries()` qui va renvoyer un objet de type Iterator contenant le couple clé/valeur pour chaque élément du tableau.

```
for (let [index, language] of languages.entries()) {
    console.log(`Index : ${index} => Valeur : ${language}`);
}
```

React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Ecmascript a aussi faciliter le travail avec les tableaux avec la fonction **map()** par exemple

```
const patronyme = item => `${item} Dalton`  
  
// we have an array  
const bandits = ['Jack', 'Avrell','Avrell', 'Jo', 'Ma\\']  
  
// call the function  
const familleArray = bandits.map(patronyme)  
console.info(familleArray)  
  
//flat Array  
const arr1 = [0, 1, 2, [3, 4]];  
  
console.log(arr1.flat());  
// expected output: [0, 1, 2, 3, 4]  
  
const arr2 = [0, 1, 2, [[[3, 4]]]];
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Les opérateurs logiques

L'opérateur OU logique va tenter de transformer la première expression en true. Si elle peut être convertie, alors cette valeur est renvoyée. Sinon c'est l'autre valeur qui est renvoyée.

```
name = "superAgent";
console.info(name || "Agent"); // "superAgent"
name = false;
console.info(name || "Agent"); // "Agent"
name = undefined;
console.info(name || "Agent"); // "Agent"
name = null;
console.info(name || "Agent"); // "Agent"
```

ES6

React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



les classes et l'héritage "facile". Comme pour un objet littéral, vous n'avez pas besoin d'utiliser le mot clé function pour déclarer une méthode.

```
class Person {  
    constructor(name, acredit) {  
        this.name = name;  
        this.acredit = acredit;  
    }  
    toString() {  
        return `Hi I'm ${this.name} and I'm the 00${this.acredit} agent !`;  
    }  
}  
let p = new Person("James Bond", 007);  
console.log(p.toString());
```



React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Opérateur ... "spread" permet de ventiler un objet dans différentes propriétés

```
var agent ={ nom : "Bond", prenom : "James"};
var equipe ={ tech : "Q", chef : "M"};
var employeur ="MI6";
```

```
var agent3 = { ...agent, employeur}
console.info(agent3)
```

```
var agent35 = { ...agent, ...employeur}
console.info(agent35)
```

```
var agent2 = {...agent,...equipe}
console.info(agent2)
```

React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Déstructuration

Ces expressions utilisant des littéraux pour les objets ou les tableaux permettent de créer simplement des données regroupées.

Une fois créées, on peut les utiliser de n'importe quelle façon

```
function destruc() {  
    return [1, 2, 3];  
}  
var [a, , b] = destruc();  
console.info("A vaut " + a + " B vaut " + b);
```

A vaut 1 B vaut 3

ES6

React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Dans chaque module on décide quelles sont les données que l'on souhaite rendre visible à l'extérieur

personnes.js

```
class Personne{  
    constructor(nom, prenom) {  
        this.nom = nom;  
        this.prenom = prenom;  
    }  
    log() {  
        console.info(` Nom : ${this.nom}, Prénom : ${this.prenom}`);  
    }  
}  
export {Personne};
```

index.html

```
</head>  
<script type="module">  
Import{Personne} from './personnes.js';  
var personne = new Personne("Bond", "James");  
Personne.log();  
</script>
```

ES6

React.JS

un framework JavaScript

Introduction

IHM

ECMASCRIPT 6



Import asynchrone

```
import Personne from "./personne.js";
import Homme from "./homme.js";

const module='./chef.js'
import(module).then((module) => {
    console.log(module);
    module.presentation();
    module.nomination();
    module.default();
});
```



React.JS

un framework JavaScript

Introduction

API navigateur

Workers

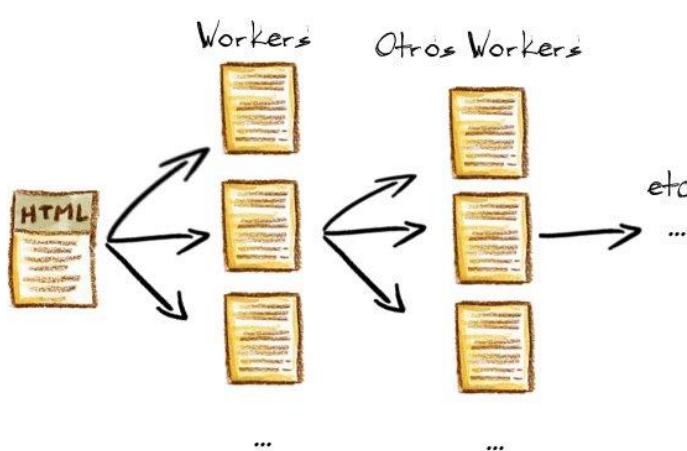


Parmi celles-ci se trouvent les **web-workers** ou la possibilité d'exécuter du code en parallèle en javascript.

Depuis sa création, le javascript souffre d'un défaut de taille : il est mono-thread !

Il existe deux types de **web-workers**. Le premier que nous allons analyser ici correspond aux **dedicated-workers** .

Ces workers, une fois créés, sont liés à leur créateur et uniquement à celui-ci.



React.JS

un framework JavaScript

Introduction

API navigateur

Websocket



On crée un worker :

- Le fichier worker.js est chargé dans un environnement qui tourne en tâche de fond (qu'on nomme souvent background), il s'agit d'un thread différent du thread principal.
- Un message est envoyé à main.js pour lui dire que le jardinier travaille
- On définit le jardin
- Dans la boucle infinie, pour chaque parcelle on fait pousser la plante
- Lorsqu'une plante est cueillie, on envoie un message indiquant le nombre de plantes cueillies.

Quand un message du worker est reçu, on l'affiche pour avoir une réponse, il faut attendre que son interlocuteur prenne le temps de lire la réponse puis envoie à son tour un message. La communication est asynchrone.

AJAX fonctionne exactement de la même manière.



React.JS

un framework JavaScript

Introduction

Ajax

Fetch



Fetch

avec XMLHttpRequest. Fetch fournit une meilleure alternative qui peut être utilisée facilement par d'autres technologies comme les Service Workers

```
const url = 'https://randomuser.me/api/?results=10';

fetch(url)
  .then((resp)=> resp.json())
  .then(function(data){
    let authors = data.results;
    let i =0;
    return authors.map(function(author){
      console.log(i);
      i++;

      let li = createNode("li"),
        img = createNode('img'),
        span = createNode('span');
      img.src=author.picture.medium;

      span.innerHTML = `${author.name.first}
${author.name.last}`;
    })
  })
  .then(function(result){
    let ul = document.createElement("ul");
    result.forEach(function(item){
      ul.appendChild(item);
    })
    document.body.appendChild(ul);
  })
  .catch(function(error){
    console.error(error);
  });
}

createNode = (tag) => {
  return document.createElement(tag);
}
```

React.JS

un framework JavaScript

Introduction

Ajax

Await



Await

L'expression await interrompt l'exécution d'une fonction asynchrone et attend la résolution d'une promesse.

Lorsque la promesse est résolue (tenue ou rompue),
la valeur est renvoyée et l'exécution de la fonction asynchrone reprend

```
function resolveAfter2Seconds(x) {
  return new Promise(resolve => {
    setTimeout(() => {
      resolve(x);
    }, 2000);
  });
}

async function f1() {
  var x = await resolveAfter2Seconds(10);
  console.info(x); // 10
}
f1();
```

React.JS

un framework JavaScript

Introduction

Ajax

Promise



Lorsque la promesse est résolue (tenue ou rompue),
la valeur est renvoyée et l'exécution de la fonction asynchrone reprend

```
function removeText (amount) {
  return new Promise(resolve => {
    function _remove (index) {
      removeCharacter().then(() => {
        if (index + 1 < amount) _remove(index + 1)
        else resolve()
      })
    }
    _remove(0)
  })
}
```

React.JS

un framework JavaScript

Introduction

Ajax

Axios



Une approche très populaire consiste à utiliser la librairie **axios**, un client HTTP basé sur les Promesses

```
function launchAxios() {
  document.querySelector("#responseLog").innerHTML = ""

  axios.get('https://sbrunet.fr/react/exemple/1-ecmascript/axiosuser.json').then(function (response) {
    document.querySelector("#responseLog").innerHTML = "<ul>"
    var responseData = response.data.data;
    for (var i = 0; i < responseData.length; i++) {
      document.querySelector("#responseLog").innerHTML +=
        "<li><img src=\"" + responseData[i].avatar + "\" />" + responseData[i].first_name + " " +
        responseData[i].last_name + "</li>";
    }
    document.querySelector("#responseLog").innerHTML += "<ul>"
  }).catch(function (error) {
    alert(error);
  }).then(function () {
    console.log("Always")
  });
}
```

React.JS

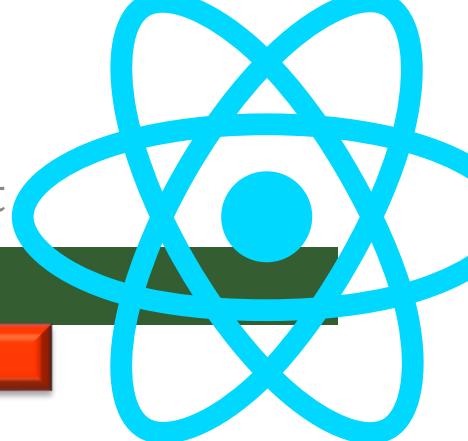
un framework JavaScript

- ❑ Rappels des composants des RIA
 - ❑ Développer avec ReactJS
 - ❑ Interactivité des composants
 - ❑ Application monopage avec ReactJS Redux
 - ❑ Application isomorphique
 - ❑ Introduction à React Native

développer patterns

développer patterns formulaire propriétés différences gestionnaire
dispatcher initiation loader logique bénéfices ecosystème comparaison pattern
développement développement côté gestion architectures suivre
extension données mise œuvre application transfert pièges routes
vie api redux dom stores états natifs
spa hot ide ins html
api ria jsx js rôle mvc os création vues css
plug virtual react ins actions ensemble
état cycle comprendre monopage détails native
contrôleurs design composant flow
render serveur choix cordova environnement reducer
transpilers applicatifs définition approche selon
node principale limitations fonctionnel

- ReactJS, positionnement et philosophie.
 - JSX, présentation. Mise en œuvre "Transpilers".
 - Approche : MVC et Virtual Dom, un choix de performance.
 - Utiliser JavaScript ou JSX.
 - Comprendre JSX en détail. Pièges à éviter.
 - Création de composant de vues. Cycle de vie.
 - Initialisation de propriétés.
 - "Render Function" : gestion des états de composant
 - Objet state



Découverte de React.JS

Framework

React.js est un framework front-end.

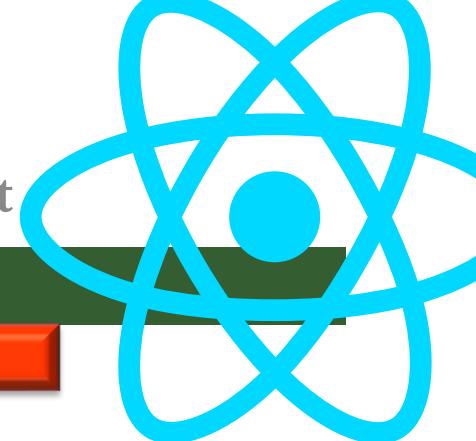
Il s'agit de tout ensemble de classes, fonctions et utilitaires qui nous **facilite la création d'applications** riches pour les navigateurs (et, de plus en plus, pour les mobiles).

Un tel framework vise à nous isoler - nous qui développons des applications web – des différences techniques entre les navigateurs, et à nous **éviter de réinventer la roue** pour tous les besoins classiques de nos applications : **gestion de l'interface utilisateur**, des événements, du DOM, des formulaires, de l'évolution dans le temps des données manipulées par l'interface, etc.

Dans cet « espace » des frameworks front-end (ou simplement « frameworks front »), on trouve pêle-mêle Angular, Ember, React, Vue.js et bien d'autres (par exemple Marko, Mavo, Backbone,...).

React.JS

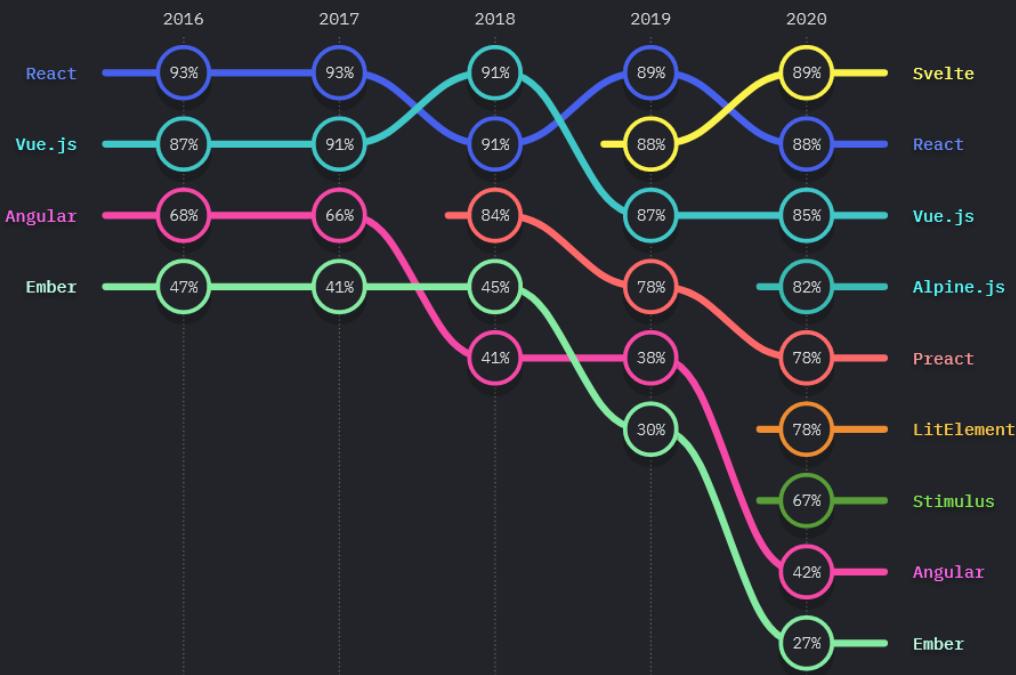
un framework JavaScript



Découverte de React.JS

Framework

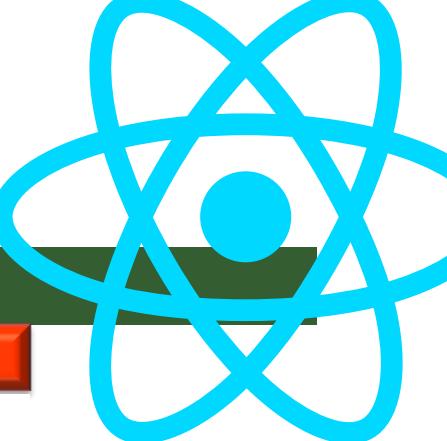
Classements des ratio de satisfaction, intérêt, utilisation et connaissance.



Résultats 2021

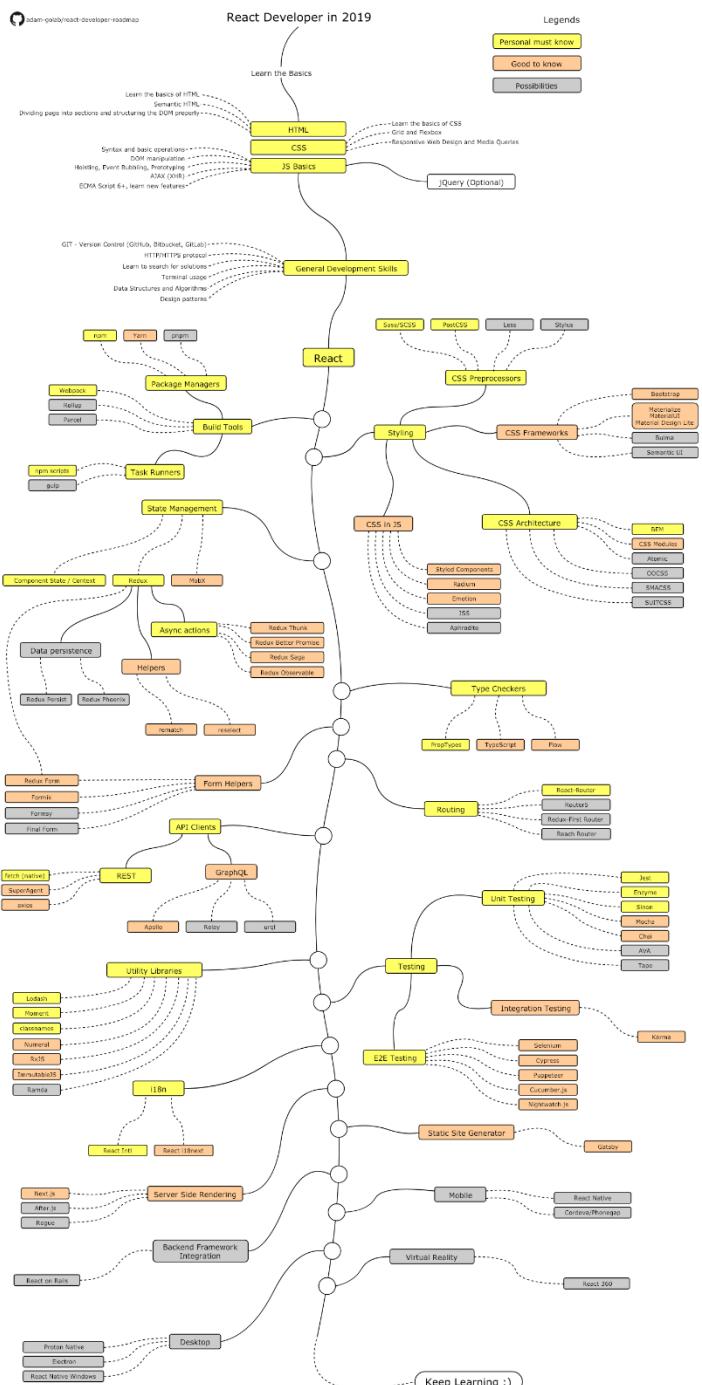
React.JS

un framework JavaScript



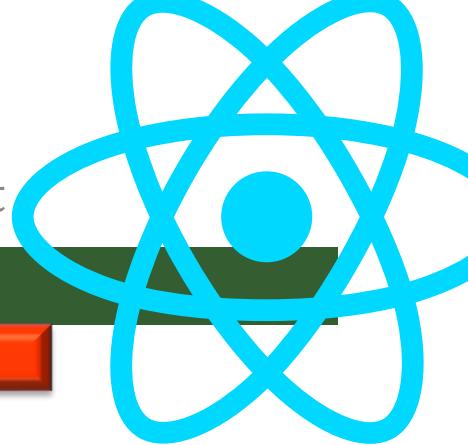
Découverte de React.JS

Framework



React.JS

un framework JavaScript



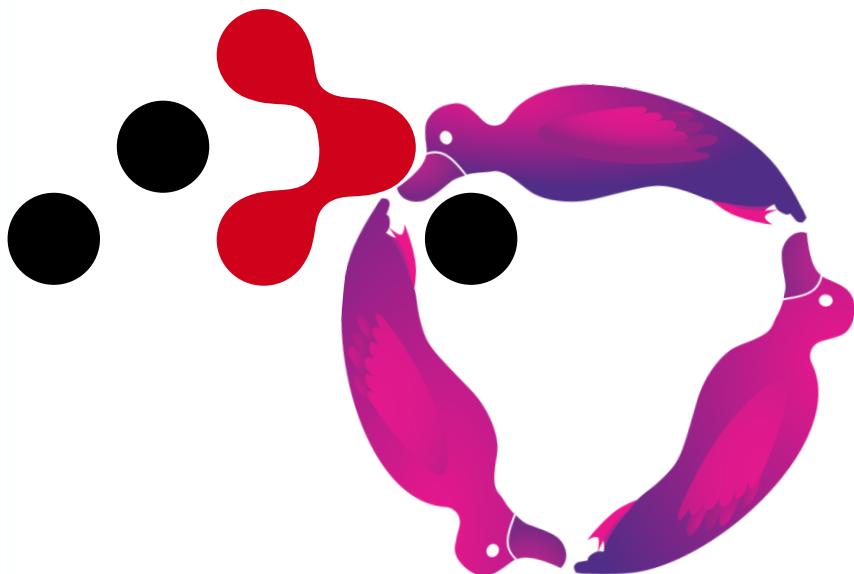
Découverte de React.JS

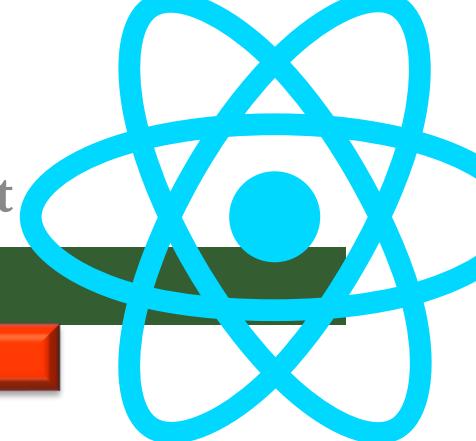
Framework

React préfère se positionner comme une bibliothèque (library) plutôt que comme un framework.

React se concentre sur le cœur du problème : la gestion de l'interface utilisateur.

Les autres briques applicatives, comme le routage côté client, le stockage des données, etc. sont laissées aux innombrables solutions complémentaires de son écosystème (par exemple, **React-Router**, **React-Redux**, **React-Native**...).





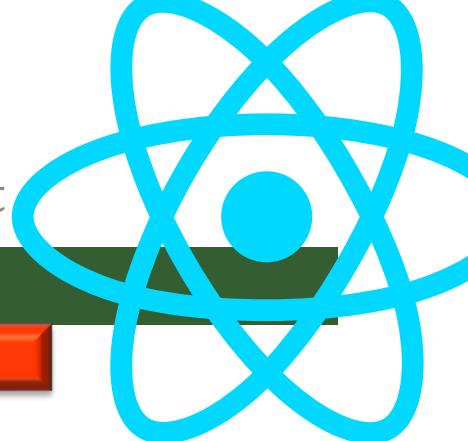
Découverte de React.JS

Framework

React est un projet **open-source**, désormais distribué sous la licence MIT et piloté par Facebook, dont tous les produits web et mobiles reposent sur cette technologie.

Autour d'une équipe principale composée d'une vingtaine de personnes, gravitent **plus de mille contributeurs** occasionnels

Le fait de rassembler dans un même fichier source ces trois volets, connectés par une **syntaxe concise** et baptisée **JSX** facilite considérablement le développement



Découverte de React.JS

Framework

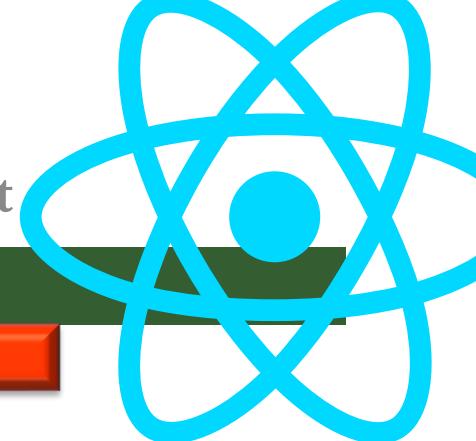
L'équipe de React pousse le concept plus loin : la version 0.14 se détache du DOM. React n'est plus une façon de représenter des éléments du DOM mais c'est une librairie qui permet de créer des composants.

Le rendu en élément DOM n'est qu'une implémentation et peut être étendu à d'autres environnements.

C'est comme ça qu'est né ReactNative qui propose d'utiliser React mais cette fois pour créer des interfaces natives pour IOS et Android. Apprendre à utiliser React vous facilite le portage de votre application sur différentes plateformes sans avoir à apprendre un nouveau langage pour chacune d'elles.

React.JS

un framework JavaScript



Découverte de React.JS

Framework

"Hello word" les premiers pas

React est composé de deux bibliothèques

React

Correspond au framework lui-même qui va nous permettre de créer des composants

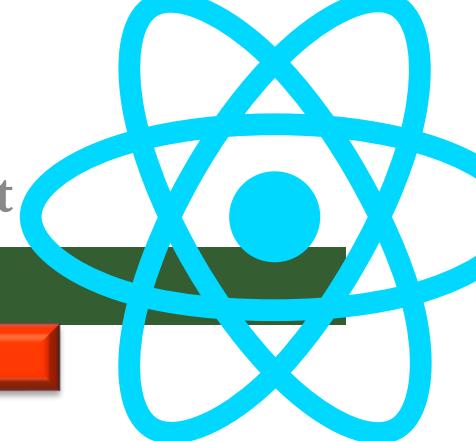
<https://unpkg.com/react@16.12/umd/react.development.js>
et,

ReactDOM une extention pour visualiser dans une page HTML

<https://unpkg.com/react-dom@16/umd/react-dom.development.js>

Dernière version 16.13 / ou 17





Découverte de React.JS

Framework

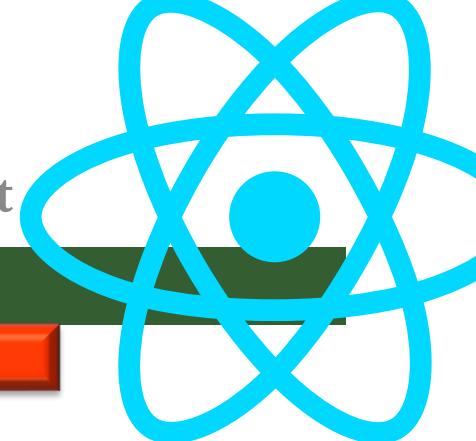
Le DOM virtuel

React a inauguré la notion de DOM virtuel : React lui-même ne manipule pas directement le DOM du navigateur.

React nous fait décrire un DOM virtuel, absolument distinct du DOM des navigateurs.

Au moment venu il réconcilie ce DOM virtuel avec la couche de rendu réelle (par exemple, le DOM du navigateur, ou, si on est côté serveur, la production du texte HTML à renvoyer côté client), en prenant soin de minimiser le nombre d'opérations nécessaires.

Ce système permet d'excellentes performances et d'utiliser React dans de nombreux contextes, et pas seulement au sein du navigateur même



Découverte de React.JS

Framework

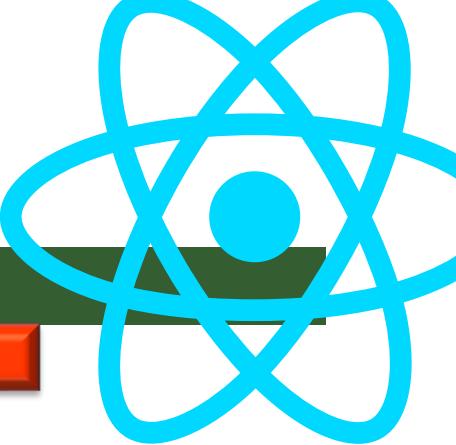
Le DOM virtuel

React a inauguré la notion de DOM virtuel : React lui-même ne manipule pas directement le DOM du navigateur.
Les deux fichiers vont être liés dans une page html

```
<html lang="fr" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>React</title>
  </head>
  <body>
    <script crossorigin src="../00-modele/js/react.development.js"></script>
    <script crossorigin src="../00-modele/js/react-dom.development.js"></script>
    <script>
      //<![CDATA[
        // ici le code React
      // ]]>
    </script>
  </body>
</html>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

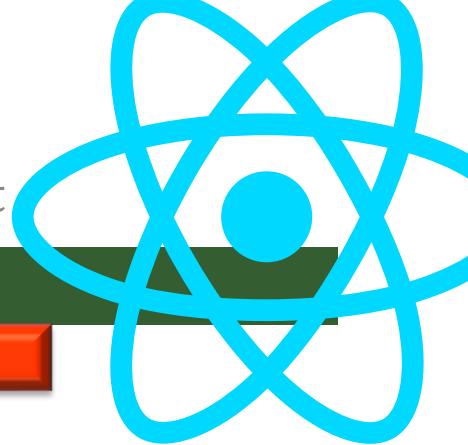
Hello world

Maintenant que les fichiers existent nous allons écrire notre premier paragraphe

```
<script>
//<![CDATA[
  var p = React.createElement("p",null,"Hello word");
  console.info(p);
  ReactDOM.render(p,document.querySelector("#app"));
// ]]>
</script>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Hello world

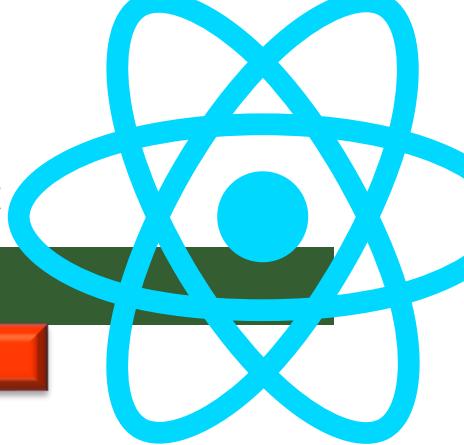
La méthode **createElement** (*élément, attributs, enfants*) permet de créer des objets

La méthode **ReactDOM.render** (*object react, element DOM ou affichage*) permet de créer des objets

```
<script>
//<![CDATA[
  var p = React.createElement("p",null,"Hello word");
  console.info(p);
  ReactDOM.render(p,document.querySelector("#app"));
// ]]>
</script>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Ajout d'attributs

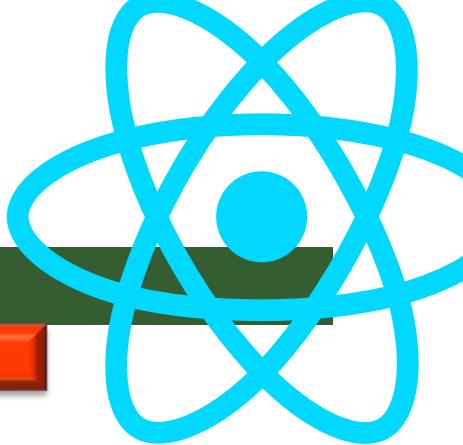
La méthode **createElement** (*élément, attributs, enfants*) permet de créer des objets

Le second paramètres permet de préciser des attributs au format JSON

```
var p = React.createElement("p", {className:"toto", id:"titi",  
  'data-test':'test'}, "Hello world");  
console.info(p.props.className);  
console.info(p.props['data-test']);  
console.info(p.props.id);  
ReactDOM.render(p, document.querySelector("#app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Ajout d'attributs

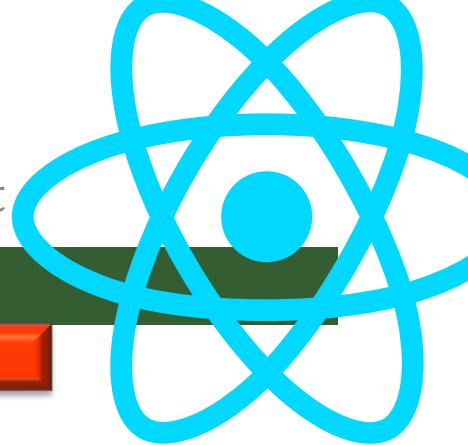
La méthode **createElement** (*élément, attributs, enfants*) permet de créer des objets

Le second paramètre permet de préciser des attributs au format JSON, on peut aussi déclarer un élément style

```
var p = React.createElement("p",
{className:"toto", style:{color:"crimson",backgroundColor:"olive"}},
  "Hello world");
// ajout d'élément CSS avec la syntaxe JS
console.info(p.props.className);
ReactDOM.render(p,document.querySelector("#app"));
// "background-color":"olive" est aussi toléré
```

"**background-color**":"olive" est aussi toléré





Découverte de React.JS

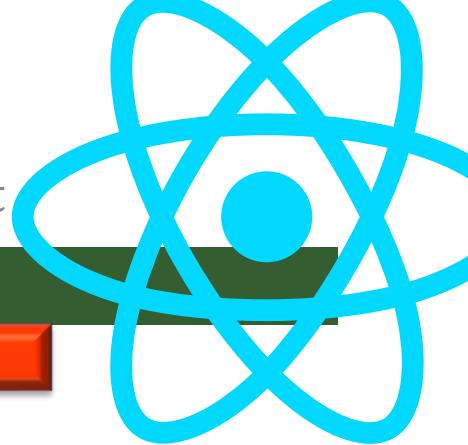
Framework

Ajout création de parent et d'enfants prenons l'exemple d'une liste

ul

li

```
//<![CDATA[  
  var ul = React.createElement("ul", {className:"liste"},  
    React.createElement("li", {className:"item"},  
      React.createElement("a", {className:"item", href:"http://www.perdu.com"},  
        "perdu")),  
    React.createElement("li", {className:"item"}, "item2"),  
    React.createElement("li", {className:"item"}, "item3"),  
    React.createElement("li", {className:"item"}, "item4"),  
    React.createElement("li", null, "item5")  
  );  
  console.info(ul);  
  ReactDOM.render(ul, document.querySelector(".app"));  
// ]]>
```



Découverte de React.JS

Framework

Insertion à partir d'une liste dynamique

Les éléments de la liste sont insérés dans le tableau 'elem'

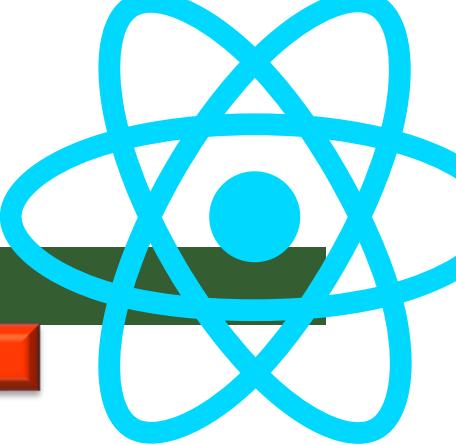
Le tableau est parcouru par la méthode *map*

```
<script>
//<![CDATA[
var elems = ["Item1","Item2","Item3","Item4","Item5",]
var ul = React.createElement("ul",{className:"liste"}, 
    elems.map(function(elem, index){
        return React.createElement("li",{className:"item"}, elem);
    })
);
console.info(ul);
ReactDOM.render(ul,document.querySelector(".app"));
// ]]>
</script>
```

Warning: Each child in a list should have a unique "key" prop. Check the top-level render call using . See <https://fb.me/react-warning-keys> for more information. in li

React.JS

un framework JavaScript



Découverte de React.JS

Framework

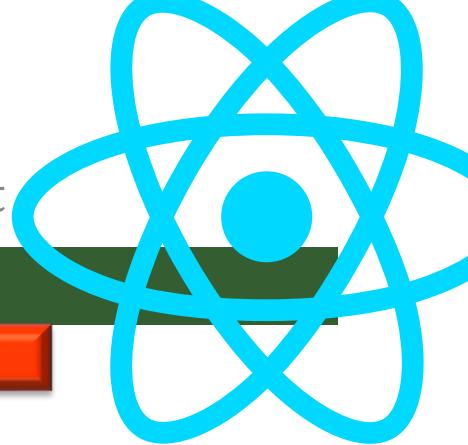
Il faut que chaque clé est un ID unique

La valeur de "key" est celle de l'index de l'élément,

```
<script>
//<![CDATA[
  var elems = ["Item1","Item2","Item3","Item4","Item5",]
  var ul = React.createElement("ul",{className:"liste"},
    elems.map(function(elem, index){
      return React.createElement("li",{className:"item",key : index}, elem);
    })
  );
  console.info(ul);
  ReactDOM.render(ul,document.querySelector(".app"));
// ]]>
</script>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

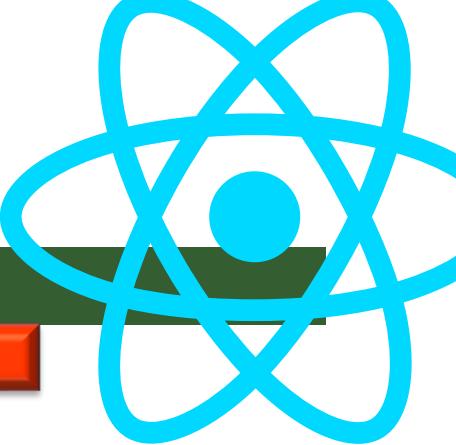
Avec ES6 on peut écrire la fonction plus simplement

```
<script>
//<![CDATA[
  var elems = ["Item1","Item2","Item3","Item4","Item5",]
// version ES6
  var ul = React.createElement("ul",{className:"liste"}, 
    elems.map((elem, index)=>{
      return React.createElement("li",{className:"item",key : index}, elem);
    })
  );
console.info(ul);
  ReactDOM.render(ul,document.querySelector(".app"));
// ]]>
</script>
```

ES6

React.JS

un framework JavaScript



Découverte de React.JS

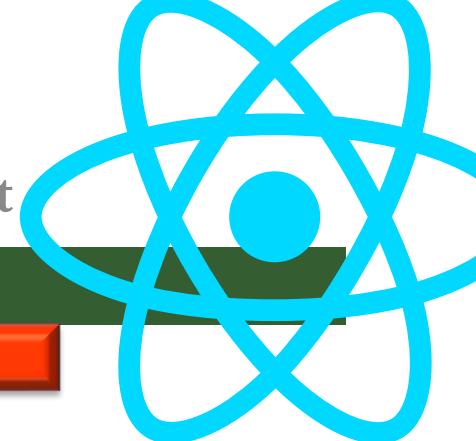
Framework

Transmettre l'attribut elems à la **fonction** listeElements()

```
//<![CDATA[  
var elems = ["Item1","Item2","Item3","Item4","Item5",]  
var listeElements = ({ elems })=>{  
    return React.createElement("ul", null,  
        elems.map((elem, index)=>{  
            return React.createElement("li", {className:"item",key : index}, elem);  
        })  
    );  
}  
var liste = React.createElement(listeElements,{elems : elems})  
ReactDOM.render(liste,document.querySelector(".app"));  
// ]]>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

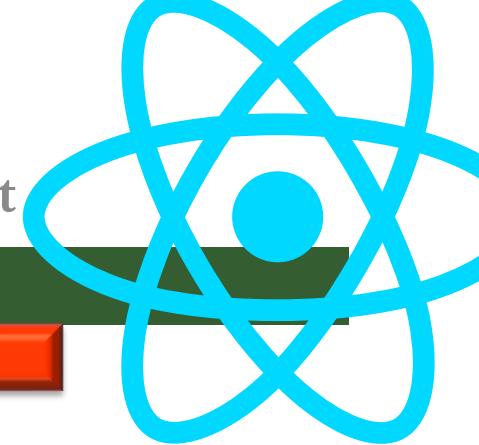
La class **React.Component**

La méthode `React.createElement` dérive d'une class `React.Component`
testons cette nouvelle **class**

```
//<![CDATA[  
    var elems = ["Item1","Item2","Item3","Item4","Item5",]  
    class listeElements extends React.Component{  
        constructor(props){  
            super(props);  
        }  
        render(){  
            return React.createElement("ul",null,  
                this.props.elems.map(function(elem, index){  
                    return React.createElement("li",{key : index}, elem);  
                })  
            );  
        }  
    }  
    var liste = React.createElement(listeElements,{elems : elems})  
    ReactDOM.render(liste,document.querySelector(".app"));  
// ]]>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

La class React.Component

La méthode React.createElement dérive d'une class React.Component
testons cette nouvelle class nous allons ajouter une classe mais nous
allons remplacer "function" par la syntaxe ES6
"=>"

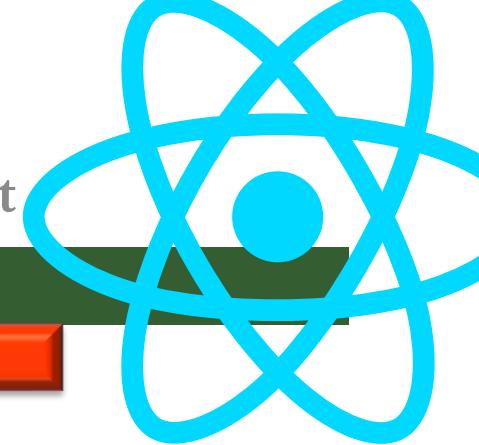
```
class listeElements extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return React.createElement("ul",null,
      this.props.elems.map((elem, index)=>{
        return React.createElement("li",{
          key : index,
          style : this.props.style
        }, elem);
      })
    );
  }
}
var liste = React.createElement(listeElements,{
  elems : elems,
  style : {color : "crimson"} //pas de this car la fonction this change de scope
});
ReactDOM.render(liste,document.querySelector(".app"));
```



ES6

React.JS

un framework JavaScript



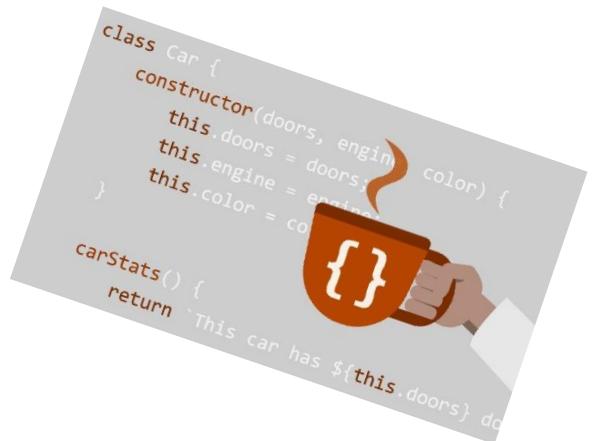
Découverte de React.JS

Framework

On peut donc soit utiliser une Class ou une fonction

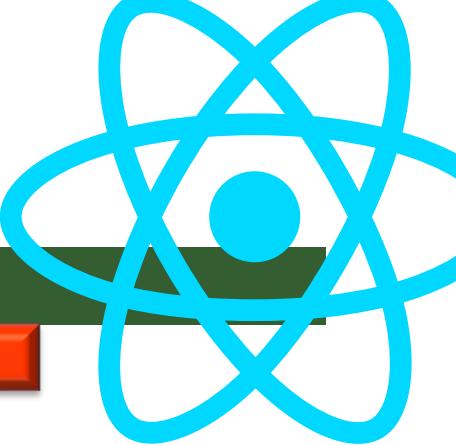
La class permet de définir des méthodes et des variables internes, que la fonction ne permet pas.

En fonction de la complexité de l'élément à créer on utilisera la fonction ou la classe



React.JS

un framework JavaScript

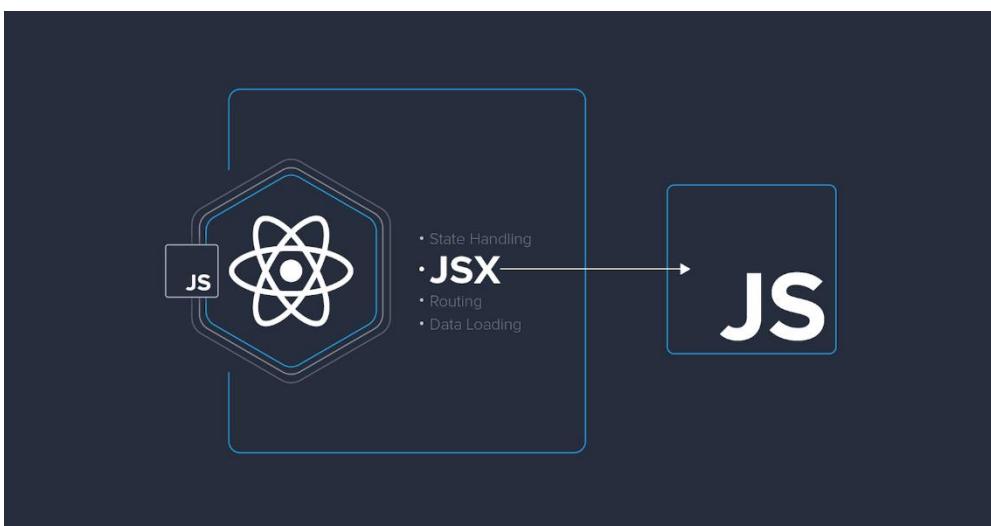


Découverte de React.JS

Framework

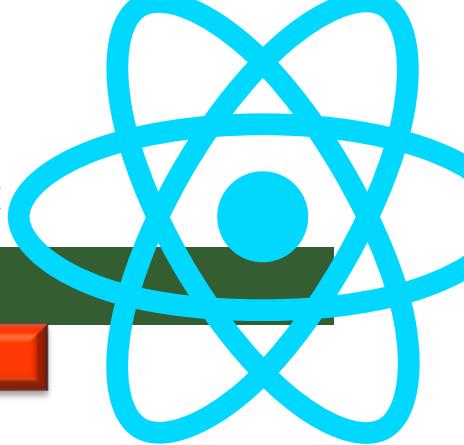
Dans le but d'alléger l'écriture du HTML au format javascript,
React propose l'utilisation de la librairie **JSX**.

Elle permet une écriture d'éléments React plus simple à lire et à écrire
que les instructions vues précédemment **React.createElement()**



React.JS

un framework JavaScript



Découverte de React.JS

Framework

Nous allons déclarer une variable avec du code jsx

```
const p = <p> hello World</p>; // code JSX  
console.info(p);  
ReactDOM.render(p, document.querySelector(".app"));
```

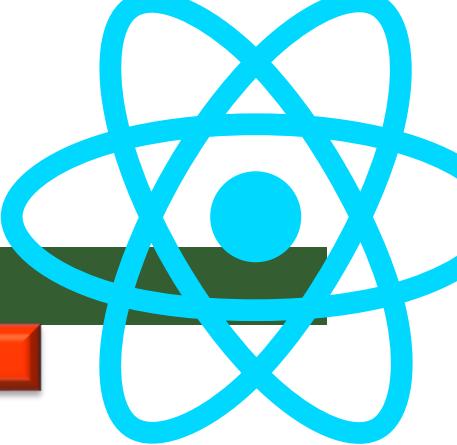
Cette variable va entraîner une erreur.

Pour résoudre il faudra traduire cette chaîne de caractères en "vrai" JavaScript on peut utiliser Babel



React.JS

un framework JavaScript



Découverte de React.JS

Framework

Nous allons déclarer une variable avec du code jsx

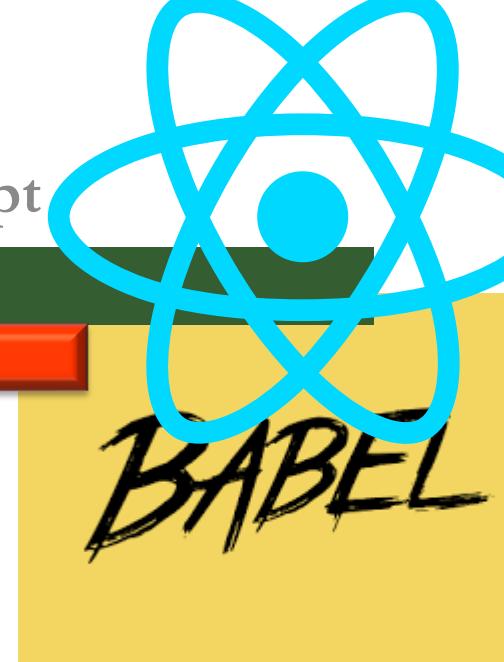
```
const p = <p> hello World</p>; // code JSX  
console.info(p);  
ReactDOM.render(p, document.querySelector(".app"));
```

Tous les navigateurs ne supportent pas toutes les propriétés de ES6.
Nous utilisons donc **Babel** à cette fin. **Babel** compile le code ES6 en
code ES5 pour qu'il puisse s'exécuter dans l'ancien navigateur.



React.JS

un framework JavaScript



Découverte de React.JS

Framework

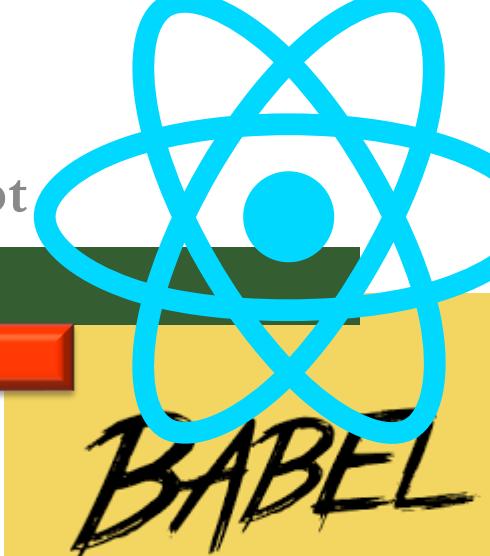
Nous allons déclarer une variable avec du code jsx

```
const p = <p> hello World</p>; // code JSX
console.info(p);
ReactDOM.render(p, document.querySelector(".app"));
```

Babel va rajouter une transcription de html en javascript
cette transcription ralenti un peu l'affichage utile surtout pour
le développement

React.JS

un framework JavaScript



Découverte de React.JS

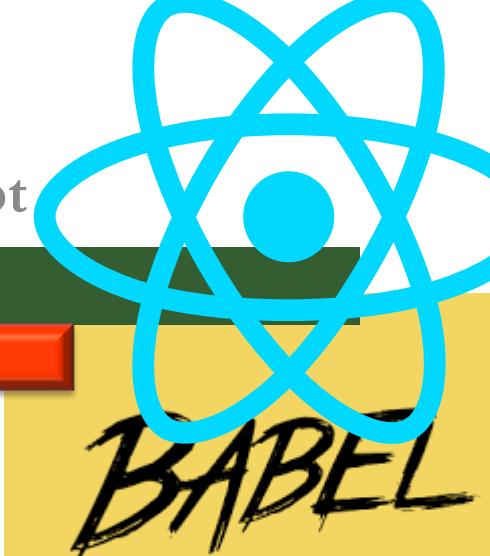
Framework

Nous allons déclarer afficher une liste

```
var liste = <ul id="list1" className="listitem">
    <li>
        <a href="http://www.perdu.com" rel="noopener">Perdu</a>
    </li>
    <li className="icon-paper-plane">Alderaan</li>
    <li>Corellia </li>
    <li>Coruscant </li>
    <li>Dagobah</li>
    <li>L{"'">}Hoth</li>
    <li>Tatooine</li>
</ul>; // code JSX
console.info(liste);
ReactDOM.render(liste, document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

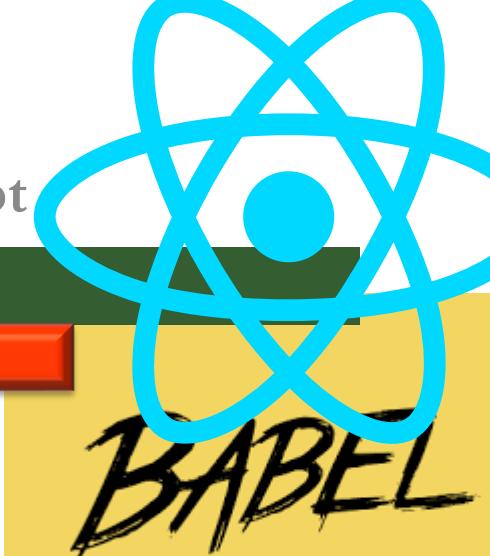
Framework

Il est possible d'ajouter des attributs style, id ...

```
var liste = <ul id="list1" className="listitem">
    <li>
        <a href="http://www.perdu.com">Item 1</a>
    </li>
    <li>Item 2</li>
    <li>Item 3</li>
    <li>Item 4</li>
    <li>Item 5</li>
</ul>; // code JSX
console.info(liste);
ReactDOM.render(liste, document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

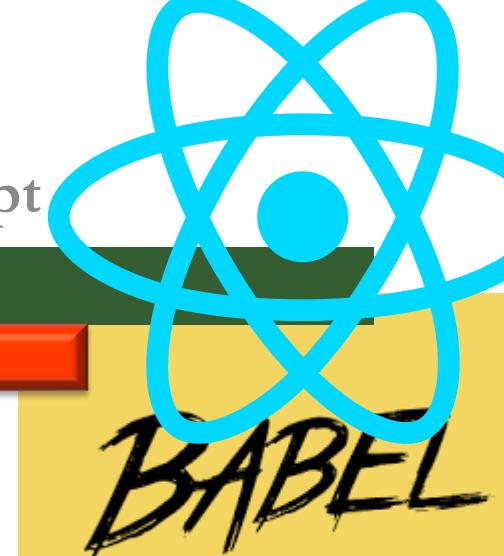
Framework

Il est possible d'ajouter des attributs style, id ...

```
var liste2 = <ul id="list2" style={{listStyleType:"none", color:"crimson"}}>
  <li>
    <a href="http://www.perdu.com">Item 1</a>
  </li>
  <li>Item 2</li>
  <li>Item 3</li>
  <li>Item 4</li>
  <li>Item 5</li>
</ul>; // code JSX
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

On peut ajouter un bloc javascript
Exemple un tableau de données
les instruction JS doivent être entourées de {}

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine"];
var color = "crimson";
var styleListe = {listStyleType:"none" , color:color};

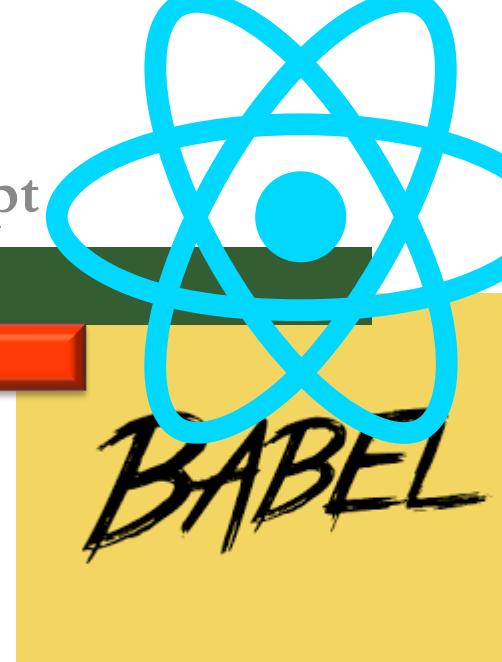
var liste = <ul id="list1" style={styleListe}>
    {elems.map((elem, index)=>// instructions JS entourées de {}
        <li key={index}>{elem}</li>
    )
}
</ul> // code JSX
console.info(liste);
ReactDOM.render(liste, document.querySelector(".app"));
```

React.JS

un framework JavaScript

Découverte de React.JS

Framework

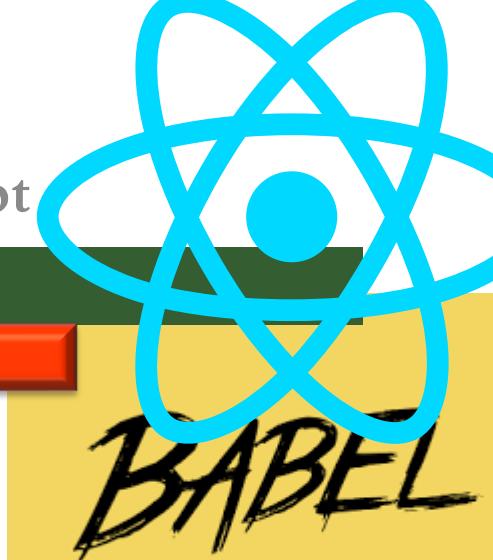


On peut ajouter un bloc javascript
Créer un élément JSX avec **une fonction**

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine" ];
var ListeElements = function(){
    // on peut aussi utiliser une fonction pour créer l'élément
    return elems.map(function(elem, index){
        return <li key={index}>{elem}</li>;
    })
}
ReactDOM.render(<ol><ListeElements /></ol>, document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

On peut ajouter un bloc javascript

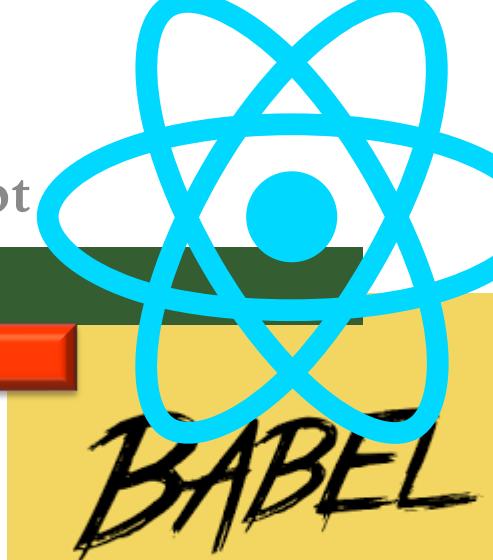
Créer un élément JSX avec une fonction transmettre des attributs dans la fonction

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine"];
```

```
//var ListeElements = function(props){  
var ListeElements = (props)=>{  
    return <ul>{  
        props.elems.map((eElem, index)=>{  
            return <li key={index} style={props.style}>{eElem}</li>;  
        })  
    }  
}</ul>  
}  
//transmettre les attributs quand on appelle la variable  
ReactDOM.render(<ListeElements elems={elems} style={{color:"crimson"}} />,  
document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

On peut ajouter un bloc javascript

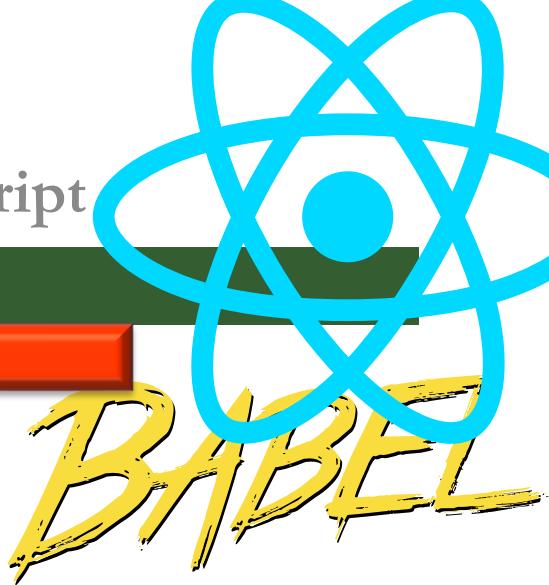
Créer un élément JSX avec une fonction transmettre des attributs dans la fonction

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine"];
```

```
//var ListeElements = function(props){  
var ListeElements = (props)=>{  
    return <ul>{  
        props.elems.map((eElem, index)=>{  
            return <li key={index} style={props.style}>{eElem}</li>;  
        })  
    }  
    </ul>  
}  
//transmettre les attributs quand on appelle la variable  
ReactDOM.render(<ListeElements elems={elems} style={{color:"crimson"}} />,  
document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

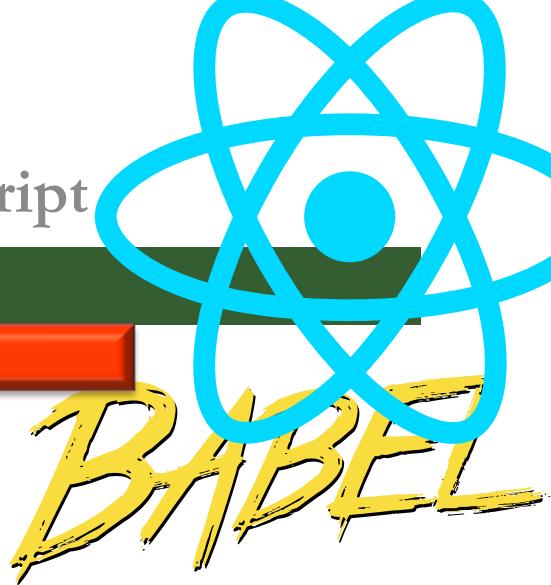
On peut ajouter un bloc javascript
Créer si les éléments sont transmises au format JSX

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine"]

//var ListeElements = function(props){
var ListeElements = (props)=>{
  return <ul>{
    props.elems.map((elem, index)=>{
      return <li key={index} style={props.style}>{elem}</li>;
    })
  }
</ul>
}
//transmettre les attributs quand on appelle la variable
ReactDOM.render(<ListeElements elems={elems} style={{color:"crimson"}} />,
  document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Utilisation de composants

React encourage à utiliser le plus de composants dans un programme

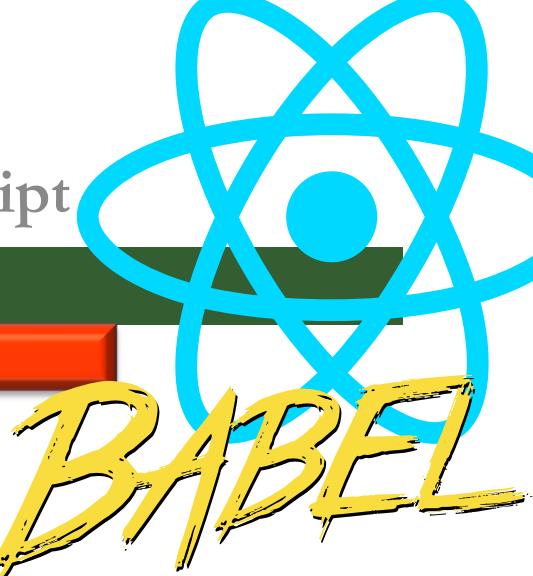
React

Composants qui pourront être réutilisé dans différentes pages

```
const elems = [ "Alderaan", "Corellia", "Coruscant", "Dagobah", "Hoth", "Tatooine"]
// création de composants réutilisables
var Element = ({color, elem})=> {
  return <li style={{color:color}}>{elem}</li>;
}
// création de composants qui peuvent appeler de composants
var ListeElements = ({elems,color})=>{
  return <ul>
    elems.map((elem, index)=>{
      return <Element key={index} elem={elem} color={color} />
    })
  </ul>
};
ReactDOM.render(<ListeElements elems={elems} color="DarkMagenta" />,
  document.querySelector(".app"));
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Utilisation de composants

Créer un élément JSX à partir d'une classe ou de plusieurs plutôt que des fonctions

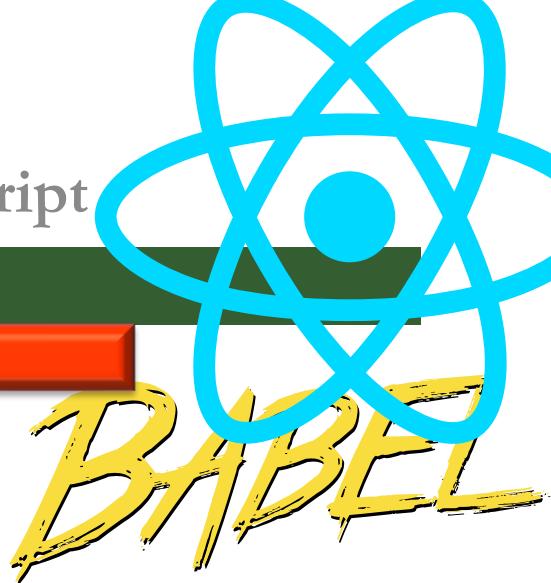
```
class Element extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return <li style={{color:this.props.color}}>{this.props.elem}</li>
  }
}
//les composants sont aussi utilisables dans les class
class ListeElements extends React.Component{
  constructor(props){
    super(props);
  }
  render(){
    return <ul>
    {
      this.props.elems.map((eElem, index)=>{
        return <Element key={index} elem={eElem}
          color={this.props.color} />
      })
    }
    </ul>
  }
}
ReactDOM.render(<ListeElements elems={elems} color="DarkMagenta" />,
```

React.JS

un framework JavaScript

Découverte de React.JS

Framework



Règles d'écriture JSX

Un seul élément parent peut être retourné

Il faut toujours un élément pour "encapsuler" tous les autres

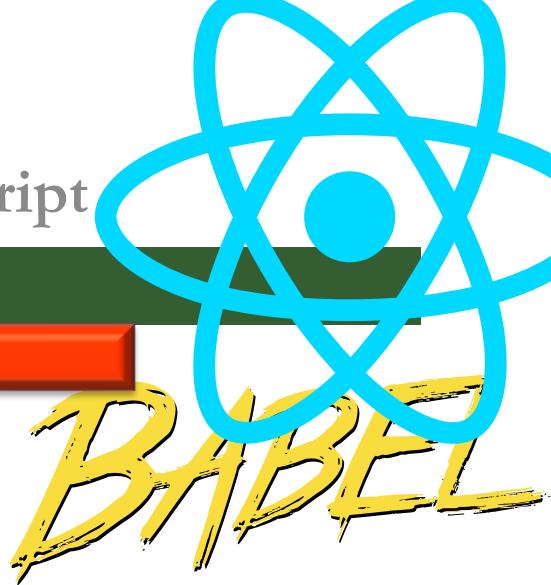
On peut utiliser React.Fragment

```
//<![CDATA[  
function ListeElements(props){  
    return <React.Fragment>  
        <article>Texte article 1 </article>  
        <article>Texte article 2 </article>  
        <article>Texte article 3 </article>  
    </React.Fragment>  
  
}  
ReactDOM.render(<ListeElements />,  
    document.querySelector(".app"));  
// ]]>
```

```
return <>  
    <aside className="frag">Athlé</aside>  
    <aside className="frag">Athlé</aside>  
    <aside className="frag">Athlé</aside>  
</>
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Règles d'écriture JSX

Un seul élément parent peut être retourné

Il faut toujours un élément pour "encapsuler" tous les autres

On peut utiliser React.Fragment

```
function ListeElements2(){
  return <>
    <aside className="frag">Athlé</aside>
    <aside className="frag">Athlé</aside>
    <aside className="frag">Athlé</aside>
    {/*<aside className="frag">Athlé</aside>} commentaire
  </>
}
```

```
ReactDOM.render(<ListeElements2/>, document.querySelector(".app2"));
```

Règles d'écriture JSX

Je commentaire // ne fonctionne en JSX

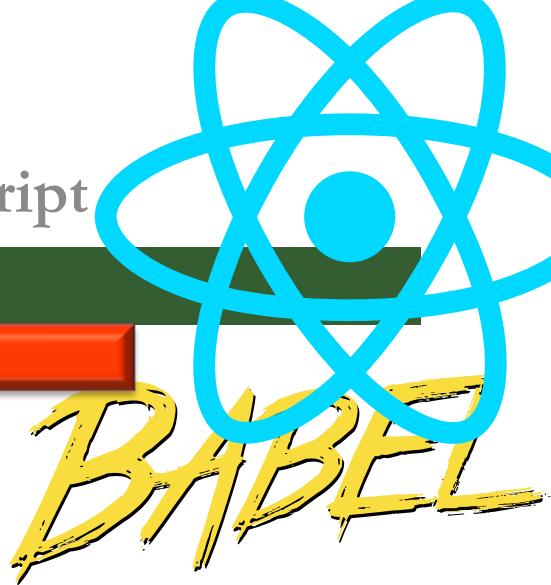
```
/* */
```

React.JS

un framework JavaScript

Découverte de React.JS

Framework



Règles d'écriture JSX

Une expression conditionnelle

```
function ListeElements(props){  
  return (<>  
    {/*affichage conditionnel*/}  
    {props.hideFirstItem ? null : <article>Coruscant</article>}  
    <article>Corellia </article>  
    <article>Tatooine</article>  
  </>)  
  
}  
ReactDOM.render(<ListeElements hideFirstItem={true} />,  
  document.querySelector(".app"));
```

```
function UserGreeting(props) {  
  return <h1>Bienvenue !</h1>;  
}  
function GuestGreeting(props) {  
  return <h1>登録してください。</h1>;  
}  
function Greeting(props) {  
  //affichage conditionnel  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <UserGreeting />;  
  }  
  return <GuestGreeting />;  
}  
ReactDOM.render(  
  // Essayez isLoggedIn={true} :  
  <Greeting isLoggedIn={false} />,  
  document.querySelector(".trad")  
>;
```

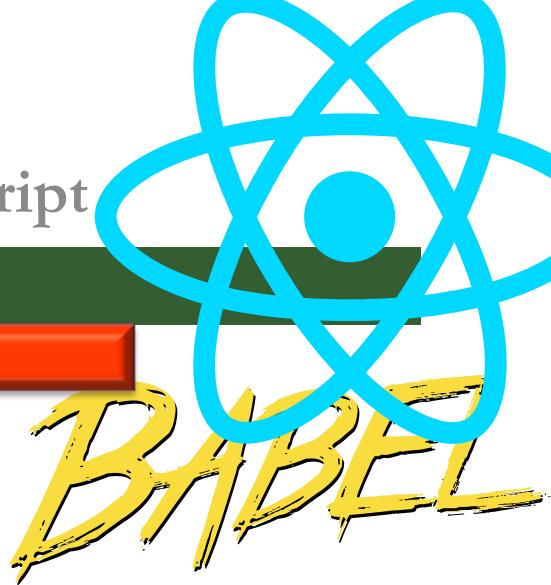
React.JS, le framework

React.JS

un framework JavaScript

Découverte de React.JS

Framework



Ajout de librairies

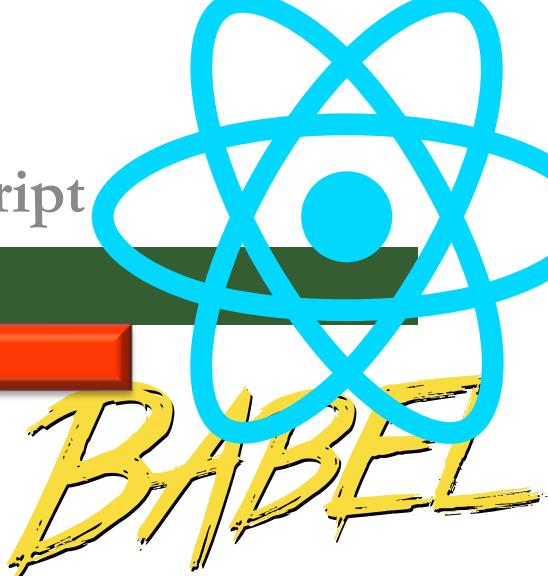
géolocalisation

```
<script src='https://unpkg.com/react-leaflet@2.3.0/dist/react-leaflet.js'></script>
<script type="text/babel">
//<! [CDATA[
const { Map: LeafletMap, TileLayer, Marker, Popup } = ReactLeaflet
class MapExample extends React.Component {
constructor() {
  super()
  this.state = {
    lat: 50.846,
    lng: 4.3472,
    zoom: 13
  }
}
render() {
  const position = [this.state.lat, this.state.lng];
  return (
    <LeafletMap center={position} zoom={this.state.zoom}>
      <TileLayer
        attribution='Nuetnigenough'
        url='https://{s}.tile.osm.org/{z}/{x}/{y}.png'
      />
      <Marker position={position}>
        <Popup>
          Nuetnigenough.be <br /> Chasseur de Geuze.
        </Popup>
      </Marker>
    </LeafletMap>
  )
}
}
</script>
```



React.JS

un framework JavaScript



Découverte de React.JS

Framework

Ajout de librairies

géolocalisation

```
<Popup>
  Nuetnigenough.be <br /> Chasseur de Geuze.
</Popup>
</Marker>
</LeafletMap>
);
}
}
ReactDOM.render(<MapExample />,
  document.querySelector(".trad")
);
```

React.JS

un framework JavaScript

Découverte de React.JS

Framework



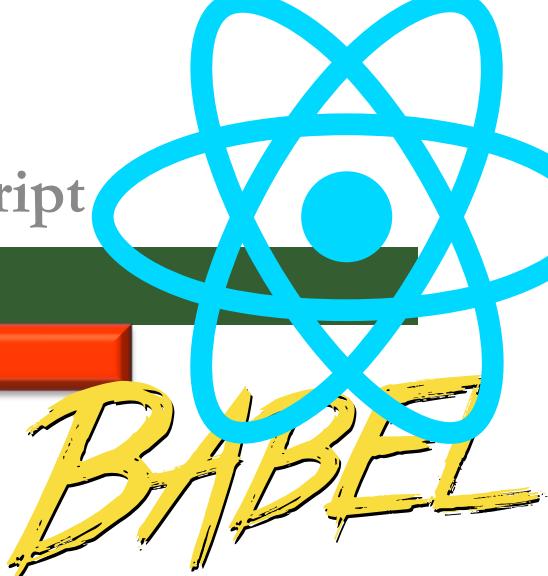
Objet state

Pour modifier l'état d'un composant il faut passer par l'objet state
States ne doit pas être modifier c'est la valeur initiale

```
class James extends React.Component {  
    // fires before component is mounted  
    constructor(props) {  
        // makes this refer to this component  
        super(props);  
        // set local state  
        this.state = {  
            name: "James"  
        };  
    }  
    render() {  
        const {name} = this.state;  
        return (  
            <div>  
                <h3>This Walter PPK is use by :</h3>  
                <Card name={name} />  
            </div>  
        )  
    }  
}  
ReactDOM.render(
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework



Objet state

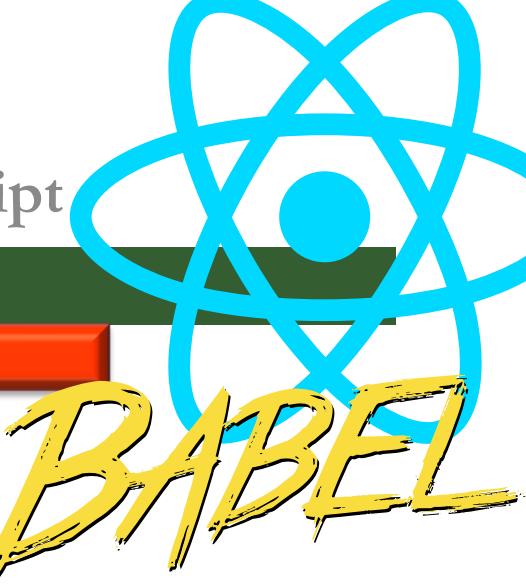
Pour modifier l'état d'un composant il faut passer par l'objet state et surtout setState

Ex mise à jour d'un composant

```
class Alarme extends React.Component{
    constructor(props){
        super(props);
        this.state={ min : 1 ,sec : 0} // creer un objet state
        setInterval(()=>{
            var newState = this.decrTime(this.state); //enlever 1sec
            console.info(newState);
            this.setState({min : newState.min, sec : newState.sec});
        }, 1000);
    }
    decrTime({min, sec}){
        sec=sec -1;
        if(sec<0){
            min=min-1;
            if (min<0){
                min = 0;
                sec = 0;
            }
        else {
            sec =59;
        }
    }
    return{min ,sec}
}
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Objet state

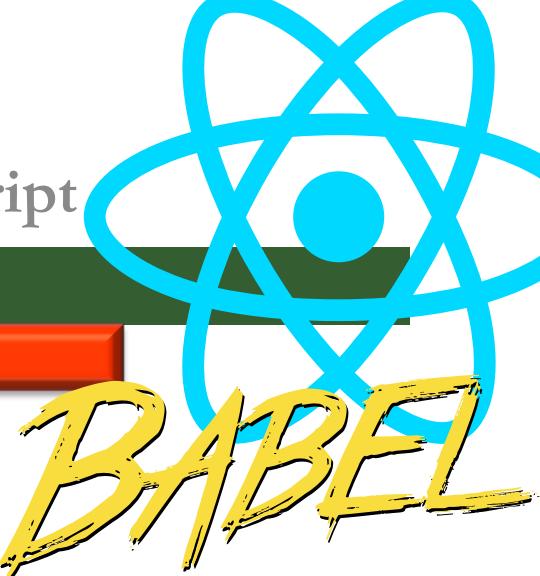
Pour modifier l'état d'un composant il faut passer par l'objet state
Ex mise a jour d'un composant

...(suite)

```
formatTime({ min, sec}){
    if (min<10) min = "0" + min;
    if (sec<10) sec = "0" + sec;
    return `${min} : ${sec}`;
}
render(){
    return <time>{this.formatTime(this.state)}</time>
}
}
ReactDOM.render(<Alarme />, document.querySelector("#timer"))
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Objet state

Pour modifier l'état d'un composant il faut passer par l'objet state
Ex mise à jour d'un composant

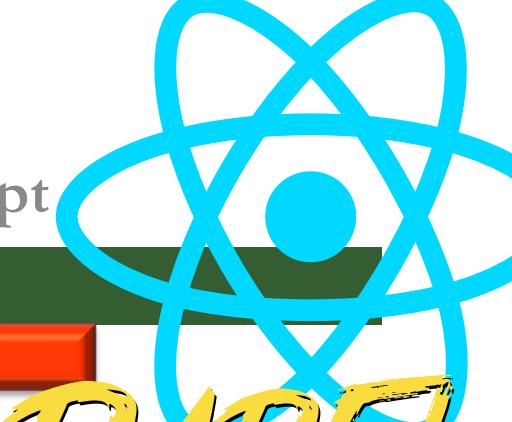
En transmettant les valeurs initiales à la classe

```
class Alarme extends React.Component{
  constructor(props){
    super(props);
    this.state={ min : props.min ,sec : props.sec} // créer un objet state
    setInterval(()=>{
      var newState = this.decrTime(this.state); //enlever 1sec
      console.info(newState);
      this.setState({min : newState.min, sec : newState.sec});
    }, 1000);
  }
  // idem au précédent
  //
  //
  ReactDOM.render(<Alarme min={2} sec={25} />,
  document.querySelector("#timer"))
  // ]]>
```



React.JS

un framework JavaScript



Découverte de React.JS

Framework

BABEL

Objet state

Pour modifier l'état d'un composant il faut passer par l'objet state

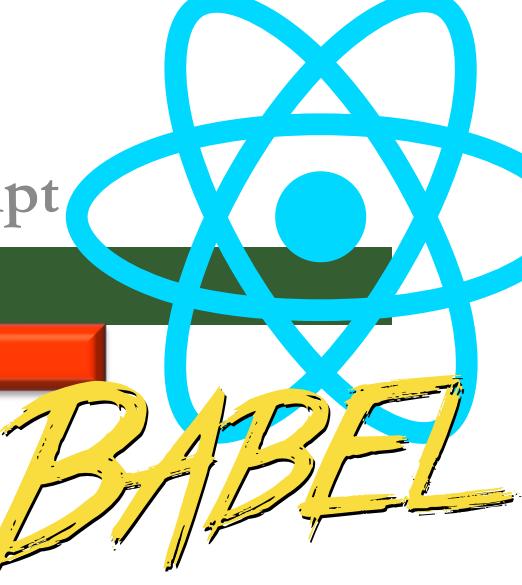
Ex mise à jour d'un composant

En transmettant les valeurs initiales à la classe

```
class Alarme extends React.Component{
    constructor(props){
        super(props);
        this.state={ min : props.min ,sec : props.sec} // créer un objet state
        setInterval(()=>{
            var newState = this.decrTime(this.state); //enlever 1sec
            console.info(newState);
            this.setState({min : newState.min, sec : newState.sec});
        }, 1000);
    }
    // idem au précédent
    render(){
        if( this.state.min==0 && this.state.sec==0){
            clearInterval(this.timer);
            return <time>Fin du temps</time>
        }
        return <time>{this.formatTime(this.state)}</time>
    }
}
ReactDOM.render(<Alarme min={2} sec={5} />,
document.querySelector("#timer")  // ])
}
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

Objet state

Cycle d'un composant à l'aide de la classe hérité de React.Component

Trois étapes :

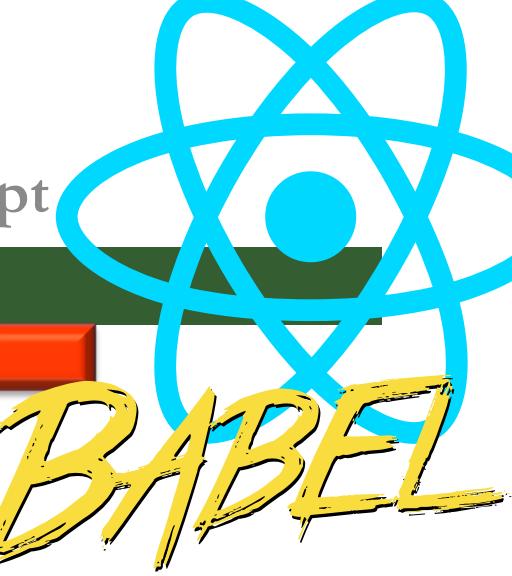
- Création du composant
- Mise a jour du composant
- Destruction du composant

Ces méthodes ne sont disponible que lors de la création d'une classe
Méthodes appelées lors de la création

- **constructor(props)** le constructeur de la class a qui on transmets les propriétés
- **componentWillmount()** : cette méthode sert a écrire le moins de code dans le constructeur
- **render()** : méthode appelée lors de l'affichage ou **this.setState()** lors de mise à jour
- **componenentDidMount()** : la méthode met le DOM à jour

React.JS

un framework JavaScript



Découverte de React.JS

Framework

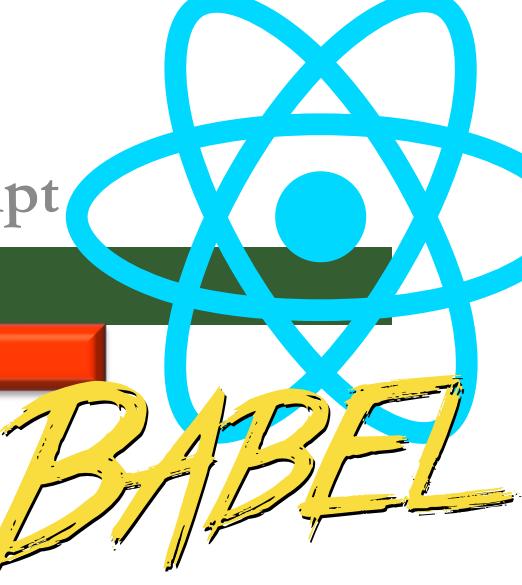
BABEL

Objet state

Cycle d'un composant à l'aide de la classe hérité de React.Component

Lors de la mise à jour les méthodes disponibles sont
Méthodes appelées lors de la mise à jour

```
class Pokemon extends React.Component {  
  constructor() {  
    super()  
    this.state = null  
  }  
  componentDidMount() {  
    fetch('https://pokeapi.co/api/v2/pokemon/wartortle/')  
      .then(res => res.json())  
      .then(res => {  
        this.setState(res)  
      })  
  }  
}
```



Découverte de React.JS

Framework

Objet state

Cycle d'un composant à l'aide de la classe hérité de React.Component

Lors de la mise à jour les méthodes disponibles sont
Méthodes appelées lors de la mise à jour

- **componentWillReceiveProps(nextProps)** : nextProps indique les propriétés transmises
- **componentWillUpdate(nextProps, nextState)** : cette méthode est appelée avant render()
- **render()** : méthode appelée lors de l'affichage ou **this.setState()** lors de mise à jour
- **componentDidUpdate(nextProps, nextState)** : une fois les composants mis à jour elles est appelée

les méthodes de cycle de vie non sécurisées étaient renommées:

componentWillMount → UNSAFE_componentWillMount

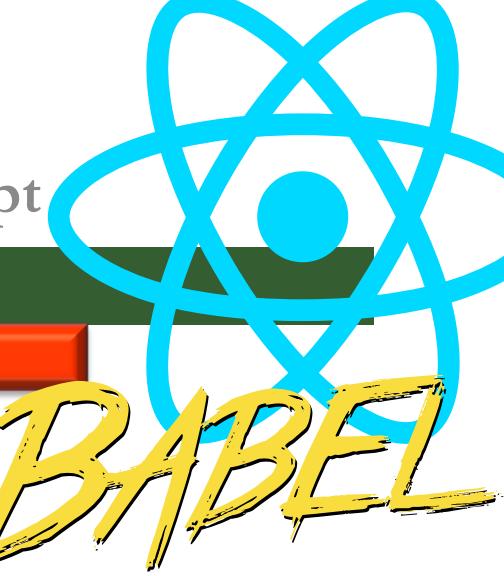
componentWillReceiveProps → UNSAFE_componentWillReceiveProps

componentWillUpdate → UNSAFE_componentWillUpdate

Deprecated

React.JS

un framework JavaScript

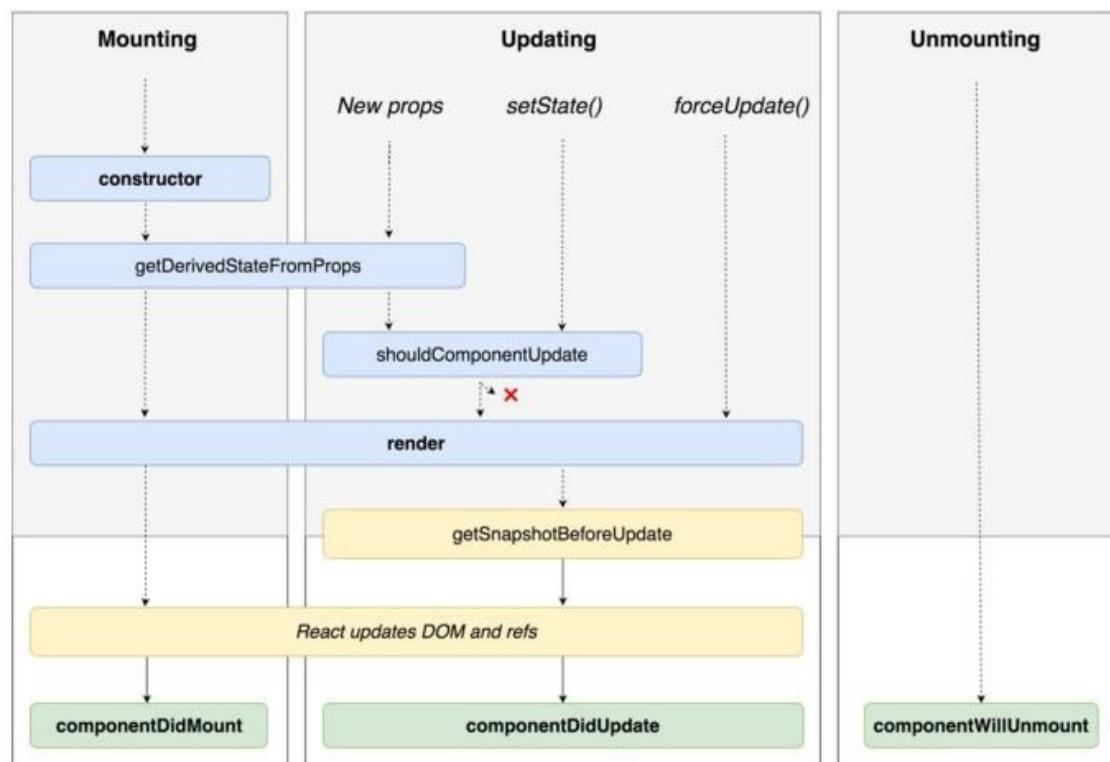


Découverte de React.JS

Framework

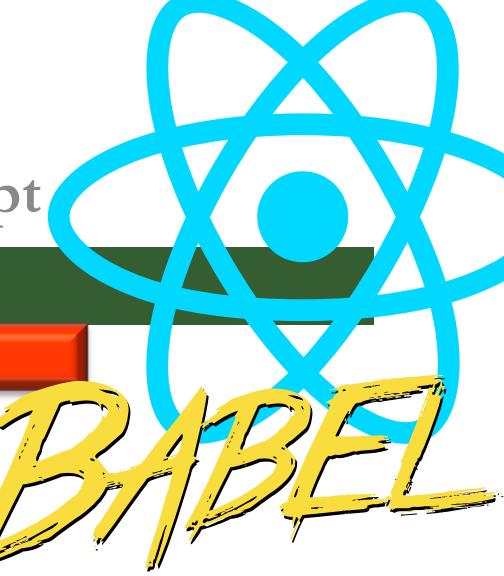
Objet state

Cycle d'un composant à l'aide de la classe hérité de React.Component



React.JS

un framework JavaScript



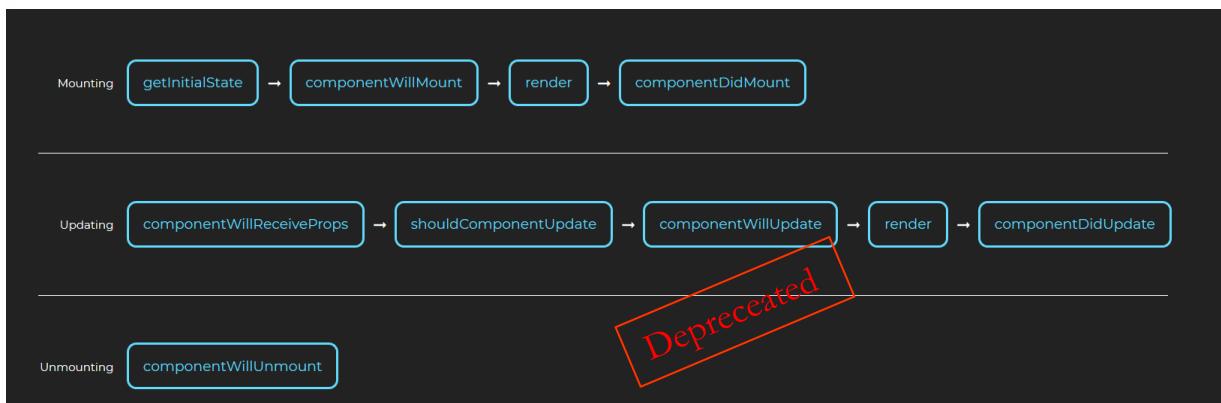
Découverte de React.JS

Framework

BABEL

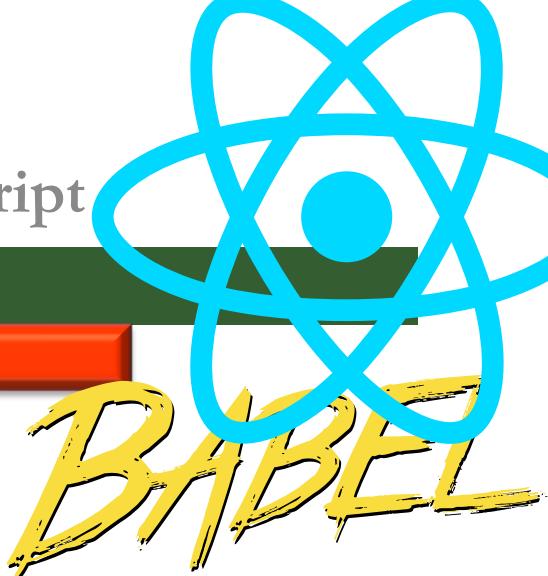
Objet state

Cycle d'un composant à l'aide de la classe hérité de React.Component



React.JS

un framework JavaScript



Découverte de React.JS

Framework



Objet state

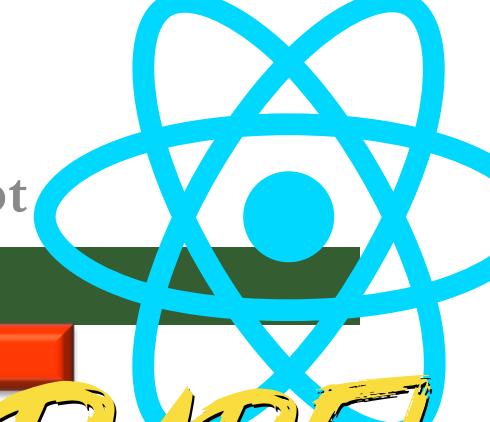
Cycle d'un composant à l'aide de la classe hérité de React.Component

```
componentWillReceiveProps(nextProps){  
  this.setState({min : nextProps.min, sec : nextProps.sec});  
  this.timer = setInterval(()=>{  
    var newState = this.decrTime(this.state);  
    console.info(newState);  
    this.setState({min : newState.min, sec : newState.sec});  
  }, 1000)  
}  
}  
ReactDOM.render(<Alarme min={0} sec={5} />,  
document.querySelector("#timer"));  
setTimeout(function(){  
  ReactDOM.render(<Alarme min={2} sec={5} />,  
  document.querySelector("#timer"));  
}, 5000);
```

Deprecated

React.JS

un framework JavaScript



Découverte de React.JS

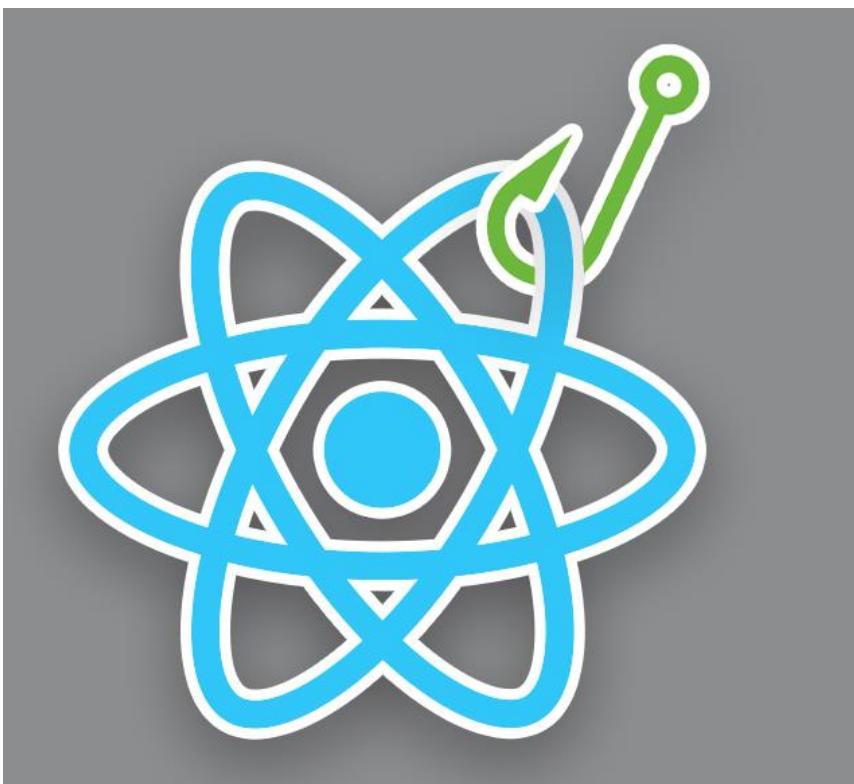
Hooks

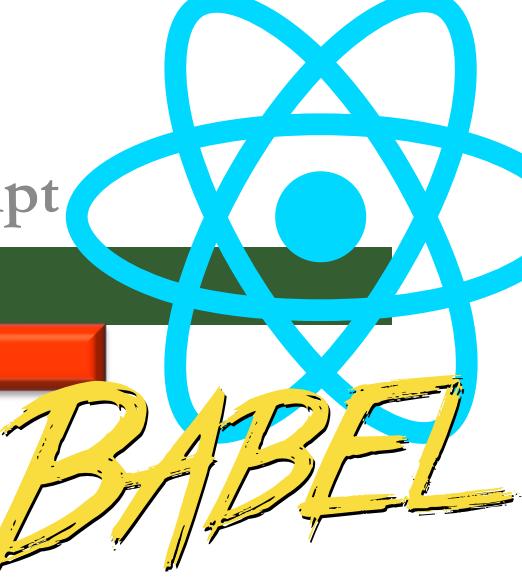
BABEL

Hooks

les hooks ont pour ambition de se passer des “**class Components**” en donnant aux “**function Components**” (nouveau nom pour les “stateless Components”).

La plupart des fonctionnalités intéressantes de React normalement réservées uniquement aux “**class Components**”.





Découverte de React.JS

Hooks

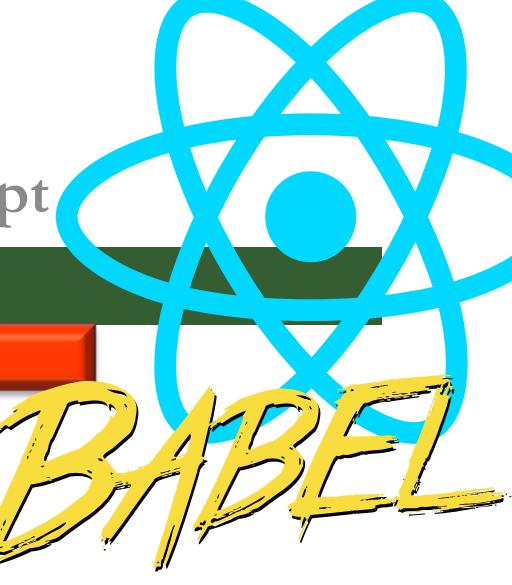
Objet state

La version 16.8 se caractérise par l'arrivée des hooks.

Les hooks permettent d'utiliser l'état ou d'autres fonctionnalités de React, sans écrire de classes.

Le développeur peut aussi construire ses propres hooks pour partager une logique dynamique entre les composants

```
function Counter() {  
    const [counter, setCounter] = React.useState(0);  
    const [error, setError] = React.useState("");  
    function increment() {  
        if (error.trim() != "") {  
            setError("");  
        }  
        setCounter(counter + 1);  
    }  
}
```



Découverte de React.JS

Hooks

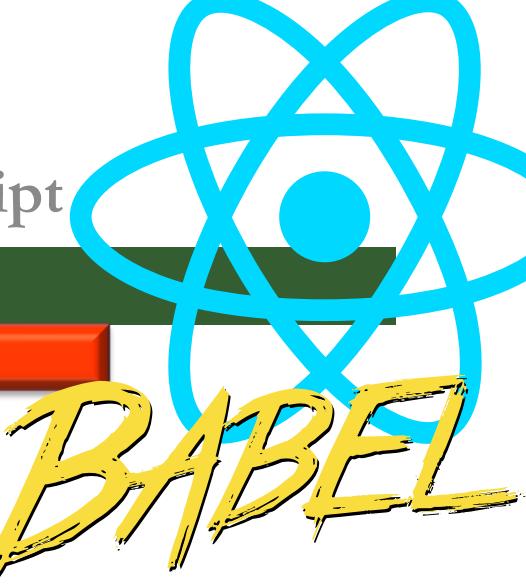
Objet state

La version 16.8 se caractérise par l'arrivée des hooks.

Les hooks permettent d'utiliser l'état ou d'autres fonctionnalités de React, sans écrire de classes.

Le développeur peut aussi construire ses propres hooks pour partager une logique dynamique entre les composants

```
const MyFirstFunctionComponent = (props) => {
  let [name, setName] = useState("");
  return (
    <input type="text" onChange={(event) =>
      setName(event.target.value)}></input>
    <span>Bonjour, je m'appelle {name}</span>
  )
}
```



Découverte de React.JS

Framework

Objet useState

Dans le code ci-dessus, **useState** est un Hook.

Nous l'appelons au sein d'une fonction composant pour y ajouter un état local.

React va préserver cet état d'un affichage à l'autre.

useState retourne une paire : la valeur de l'état actuel et une fonction qui vous permet de la mettre à jour.

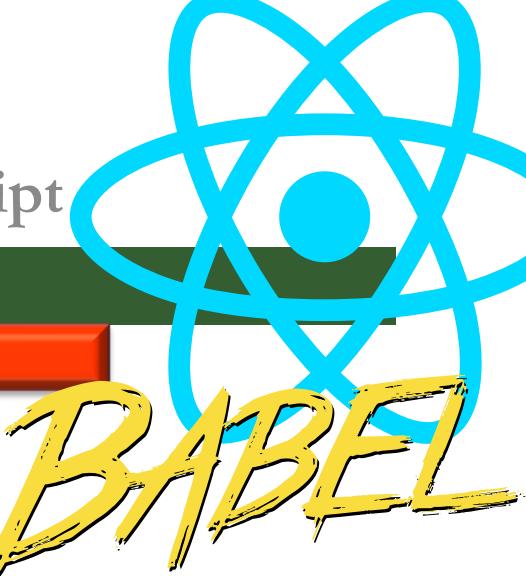
Vous pouvez appeler cette fonction depuis un gestionnaire d'événements, par exemple.

Elle est similaire à **this.setState** dans une classe, à ceci près qu'elle ne fusionne pas l'ancien état et le nouveau.

```
//initialise toutes mes const
//React peut regrouper plusieurs appels setState () en une seule mise à jour
const [state, setState] = React.useState(initialState);
const increment = () => {
  if (state.error.trim() != "") {
    setState(prev => ({
      ...prev,
      error: ''
      //ne met à jour que 'error'
    }));
  }
}
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

```
class Chart extends Component {
  state = {
    data: null,
  }

  componentDidMount() {
    const data = getDataWithinRange(this.props.dateRange)
    this.setState({ data })
  }

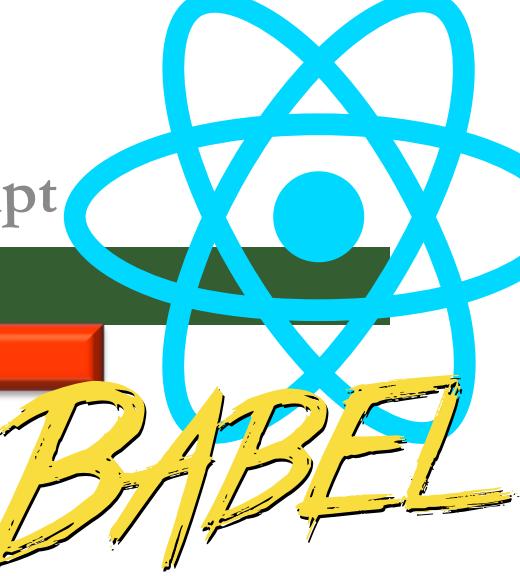
  componentDidUpdate(prevProps) {
    if (prevProps.dateRange != this.props.dateRange) {
      const data = getDataWithinRange(this.props.dateRange)
      this.setState({ data })
    }
  }

  render() {
    return (
      <svg className="Chart" />
    )
  }
}
```

```
const Chart = ({ dateRange }) => {
  const [data, setData] = useState()

  useEffect(() => {
    const data = getDataWithinRange(dateRange)
    setData(data)
  }, [dateRange])

  return (
    <svg className="Chart" />
  )
}
```



Découverte de React.JS

Framework

Objet useEffect

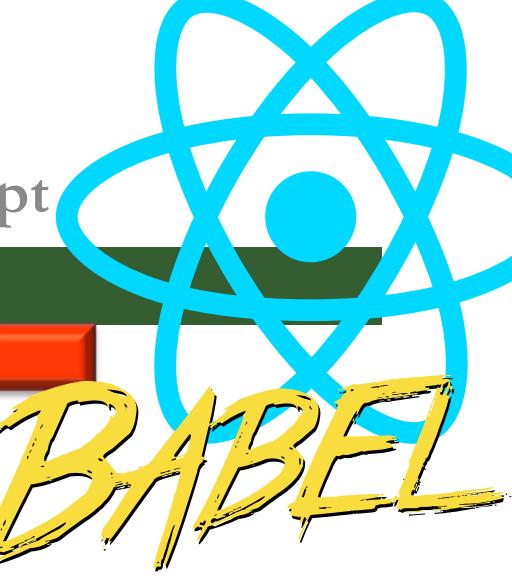
```
useEffect(didUpdate);
```

Accepte une fonction qui contient du code impératif, pouvant éventuellement produire des effets.

L'utilisation de mutations, abonnements, horloges, messages de journalisation, et autres effets de bord n'est pas autorisée au sein du corps principal d'une fonction composant (qu'on appelle la *phase de rendu* de React)

La fonction fournie à **useEffect** sera exécutée après que le rendu est apparu sur l'écran.

Vous pouvez considérer les effets comme des échappatoires pour passer du monde purement fonctionnel de React au monde impératif.



Découverte de React.JS

Framework

Objet useEffect

```
useEffect(didUpdate);
```

Permet d'injecter dans nos fonctions components la notion de lifeCycle hook utilisée dans tous les frameworks front-end récents.

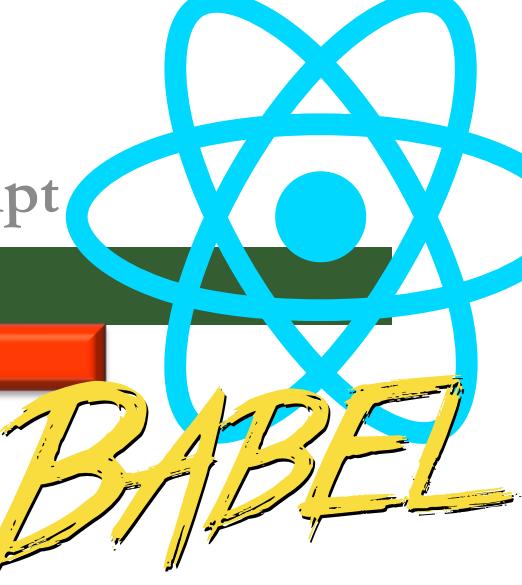
En React, il y en a plusieurs, on les connaît sous le nom de **ComponentWillMount**, **ComponenentWillReceiveProps**, **ComponentWillUpdate**, etc...

On se sert principalement de ces fonctions pour récupérer des données (call Api), mettre à jour des variables ou encore faire des modifications sur le DOM.

useEffect nous permet de combiner 3 lifecycle hooks de React, **componentDidMount**, **componentDidUpdate** et **componentWillUnmount**.

React.JS

un framework JavaScript



Découverte de React.JS

Framework

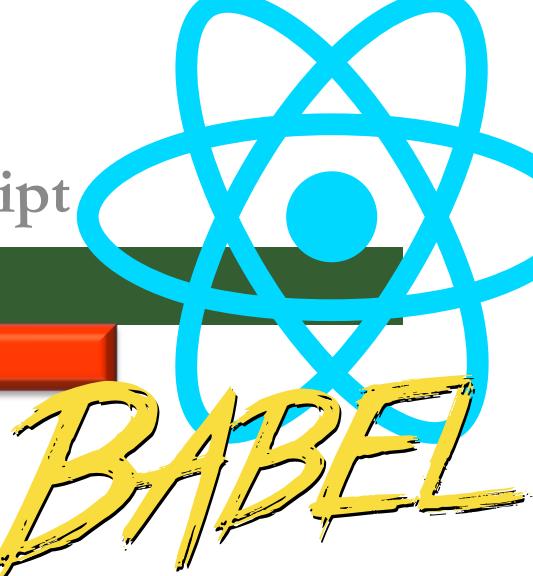
Objet useEffect

```
useEffect(didUpdate);
```

```
const { React, ReactDOM } = window
    const { useEffect, useState, Fragment } = React
    const { render } = ReactDOM
    const rootNode = document.querySelector("#job")
    const getDimensions = () => ({
        height: innerHeight,
        width: innerWidth,
    })
    const App = () => {
        const [dimensions, setDimensions] =
useState(getDimensions())
        const [x, setX] = useState()
        const [y, setY] = useState()
        const updateSize = () => setDimensions(getDimensions())
        const update = e => {
            setX(e.x)
            setY(e.y)
        }
    }
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

BABEL

CLASS COMPONENT

With class components, we tie updates to specific **lifecycle events**.

```
class Chart extends Component {
  componentDidMount() {
    // when Chart mounts, do this
  }

  componentDidUpdate(prevProps) {
    if (prevProps.data == props.data) return
    // when data updates, do this
  }

  componentWillUnmount() {
    // before Chart unmounts, do this
  }

  render() {
    return (
      <svg className="Chart" />
    )
  }
}
```

FUNCTION COMPONENT

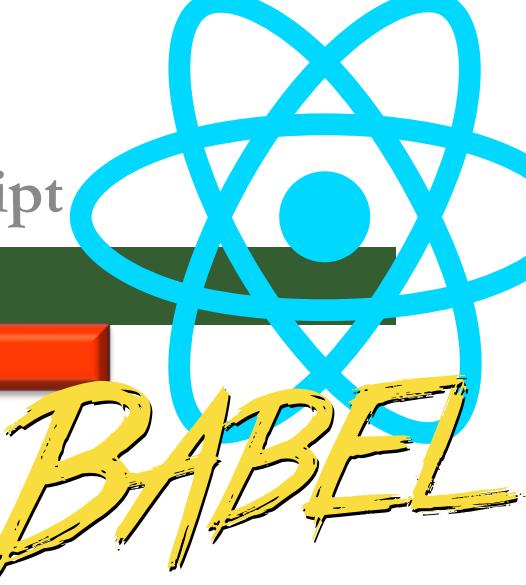
In a function component, we instead use the `useEffect` hook to run code during the major **lifecycle events**.

```
const Chart = ({ data }) => {
  useEffect(() => {
    // when Chart mounts, do this
    // when data updates, do this
  }, [data])

  return (
    <svg className="Chart" />
  )
}
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

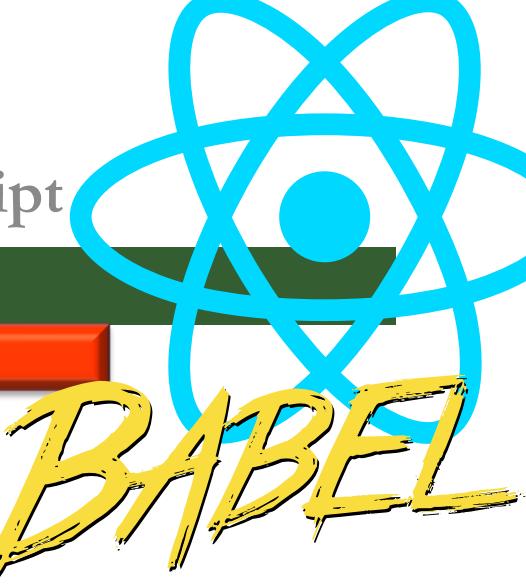
BABEL

Objet useContext

Le hook prend en paramètre un object Context précédemment créé avec `React.createContext()` et retourne la valeur du context en question.

Un composant utilisant `useContext` déclenchera un nouveau rendu lorsque la valeur du context changera.

```
const NumberContext = React.createContext();
function App() {
  return (
    <NumberContext.Provider value={42}>
      <div>
        <Display />
      </div>
    </NumberContext.Provider>
  );
}
```



Découverte de React.JS

Framework

Objet useMemo

Ce hook permet de mémoriser une valeur afin d'améliorer les performances de notre composant.

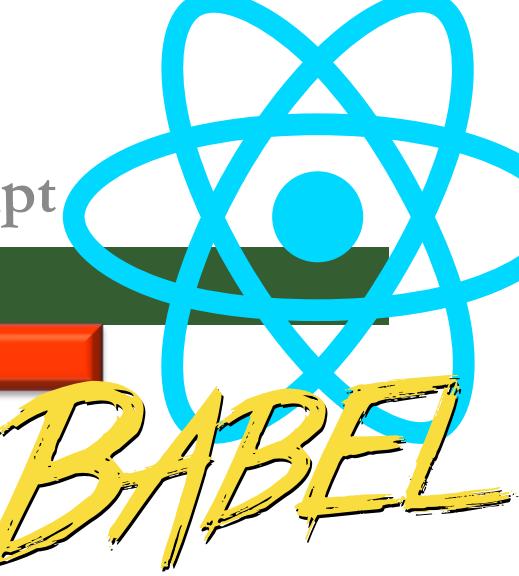
Ce hook prend en premier paramètre la fonction de création de la valeur et en deuxième argument un tableau de valeurs.

La fonction sera exécutée si et seulement si au moins une valeur du tableau change.

Cela permet empêcher les calculs inutiles et coûteux entre chaque rendu.

```
const { React, Reaconst CountButton = React.memo(function
CountButton({onClick, count}) {
return <button onClick={onClick}>{count}</button>
})
function DualCounter() {
const [count1, setCount1] = React.useState(0)
const increment1 =
React.useCallback(() => setCount1(c => c + 1), [])
const [count2, setCount2] = React.useState(0)
const increment2 =
React.useCallback(() => setCount2(c => c + 1), [])
ctDOM } = window
```





Découverte de React.JS

Framework

Objet useMemo

Renvoie une valeur mémoisée

Fournissez une fonction de « création » et un tableau d'entrées.

useMemo recalculera la valeur mémoisée seulement si une des entrées a changé.

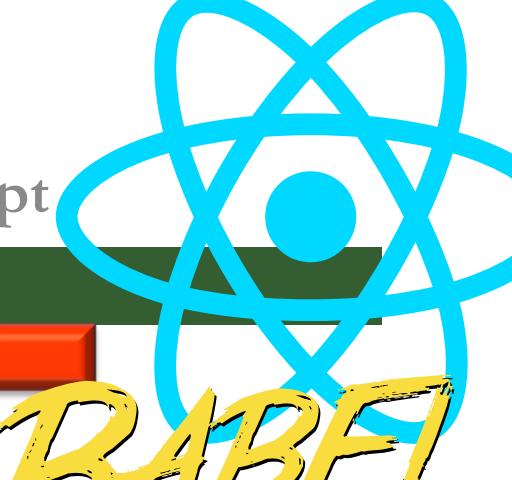
Cette optimisation permet d'éviter des calculs coûteux à chaque rendu. Rappelez-vous que la fonction fournie à useMemo s'exécute pendant le rendu.

N'y faites rien que vous ne feriez pas normalement pendant un rendu. Par exemple, les effets de bord doivent passer par useEffect, et non useMemo.

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` fait référence au champ textuel monté dans le DOM
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Donner le focus au champ</button>
    </>
  );
}
```

React.JS

un framework JavaScript



Découverte de React.JS

Framework

BABEL

CLASS COMPONENT

```
class Chart extends Component {
  state = {
    data: null,
  }

  componentDidMount() {
    const data = getDataWithinRange(this.props.dateRange)
    this.setState({ data })
  }

  componentDidUpdate(prevProps) {
    if (prevProps.dateRange !== this.props.dateRange) {
      const data = getDataWithinRange(this.props.dateRange)
      this.setState({ data })
    }
  }

  render() {
    return (
      <svg className="Chart" />
    )
  }
}
```

FUNCTION COMPONENT

In fact, this last example was still thinking inside the class-component box. We're keeping a calculated value in our `state` in order to prevent re-calculating it every time our component updates.

But we no longer need to use `state` ! Here to the rescue is `useMemo()`, which will only re-calculate `data` when its **dependency array** changes.

```
const Chart = ({ dateRange }) => {
  const data = useMemo(() => (
    getDataWithinRange(dateRange)
  ), [dateRange])

  return (
    <svg className="Chart" />
  )
}
```

React.JS

un framework JavaScript

Découverte de React.JS

Framework



```
class Chart extends Component {
  state = {
    dimensions: null,
    xScale: null,
    yScale: null,
  }

  componentDidMount() {
    this.setState({dimensions: getDimensions()})
    this.setState({xScale: getXScale()})
    this.setState({yScale: getYScale()})
  }

  componentDidUpdate(prevProps, prevState) {
    if (prevProps.margins != this.props.margins) {
      this.setState({dimensions: getDimensions()})
    }
    if (prevProps.data != this.props.data) {
      this.setState({xScale: getXScale()})
      this.setState({yScale: getYScale()})
    }
  }

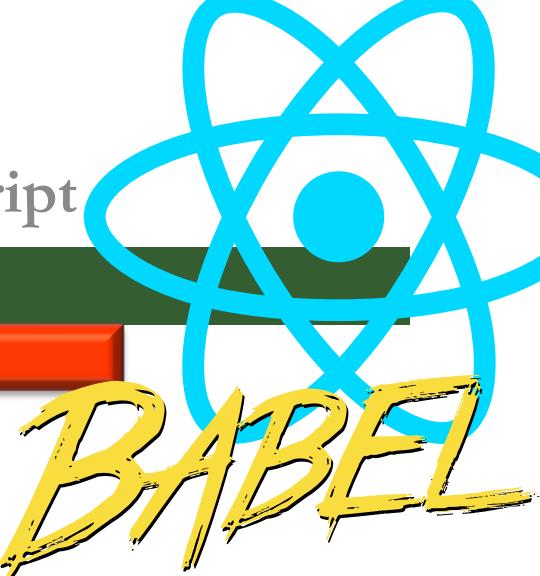
  render() {
    return (
      <svg className="Chart" />
    )
  }
}
```

```
const Chart = ({ data }) => {
  const dimensions = useMemo(getDimensions, [margins])
  const xScale = useMemo(getXScale, [data])
  const yScale = useMemo(getYScale, [data])

  return (
    <svg className="Chart" />
  )
}
```

React.JS

un framework JavaScript



Découverte de React.JS

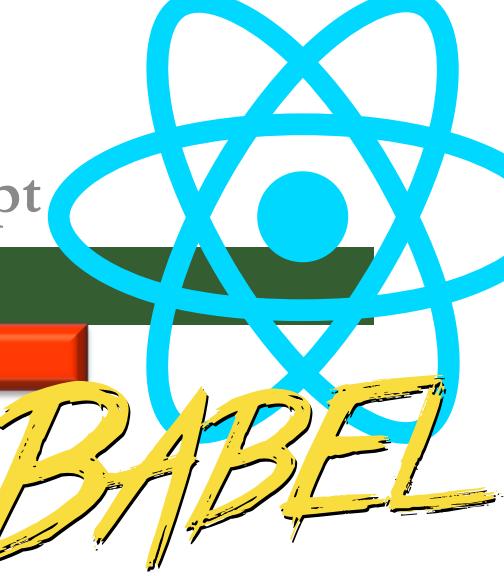
Framework

Objet useRef

useRef renvoie un objet ref modifiable dont la propriété current est initialisée avec l'argument fourni (initialValue).

L'objet renvoyé persistera pendant toute la durée de vie composant. La seule différence entre useRef() et la création manuelle d'un objet {current: ... }, c'est que useRef vous donnera le même objet à chaque rendu.

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` fait référence au champ textuel monté dans le DOM
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Donner le focus au champ</button>
    </>
  );
}
```



Découverte de React.JS

Hook

BABEL

custom Hooks

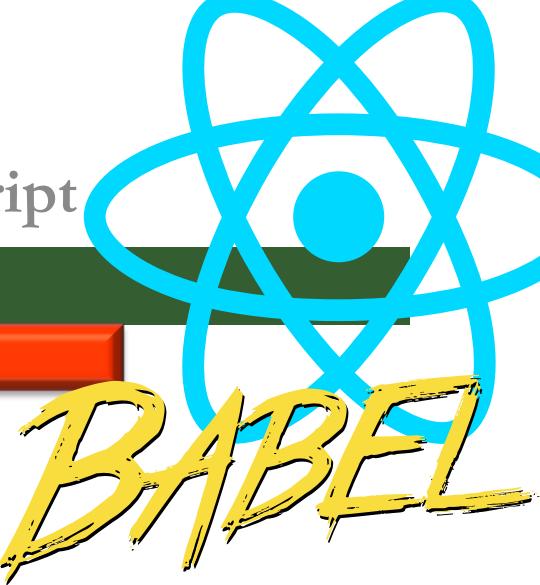
Un hook étant une fonction Javascript avec un nom commençant par use et pouvant utiliser d'autres hooks, on peut créer ses propres hook

```
function useAsyncHook(searchBook) {  
  const [result, setResult] = React.useState([]);  
  const [loading, setLoading] = React.useState("false");
```



React.JS

un framework JavaScript



Découverte de React.JS

Hook

BABEL

Objet useEffect

```
useEffect(didUpdate);
```

<https://blogs.infinitesquare.com/posts/web/les-hooks-dans-react-presentati>

<https://dev.to/finallynero/hooks-introduced-in-react-router-v5-1-7g8>

<https://dev.to/vinodchauhan7/react-hooks-with-async-await-1n9g>

<https://blog.bitsrc.io/why-we-switched-to-react-hooks-48798c42c7f>

<https://codeburst.io/react-hooks-recipes-1c18e5984abe>



React.JS

un framework JavaScript

- ❑ Rappels des composants des RIA
 - ❑ Développer avec ReactJS
 - ❑ Interactivité des composants
 - ❑ Application monopage avec ReactJS Redux
 - ❑ Application isomorphique
 - ❑ Introduction à React Native

développer pattern :

développer patterns formulaire différences gestionnaire
dispatcher initiation propriétés pattern
loader logique bénéfices comparaison
logique extension mise œuvre transfert côté gestion
bdd données application pièges routes
côté œuvre transfert côté gestion
api redux dom stores états natifs
vie ria spa hot ide ins html
api jsx js ide ins html
virtual react rôle mvc os création vues css
plug plug rôle mvc os création vues css
état état cycle comprendre monopage ensemble
cycle contrôleur design composant détail native
composants serveur render choix flow
design serveur render choix flow
contrôleur transpilers définition cordova environnement
transpilers applicatifs node principe approche selon
applicatifs limitations fonctionnal

- Gestion des événements. "autobinding" et délégation.
 - Design Pattern : stratégie pour les composants à état.
 - Composer par ensembles.
 - "Component Data Flow" : propriétaire, enfants et création dynamique.
 - Composants réutilisables : contrôle et transfert de propriétés.
 - Contrôle des composants de formulaire.
 - Manipulation du DOM.

Interactivité des composants

Gestion des événements

Click

Nous allons maintenant "surveiller" les événements et déclencher des actions

React a construit dans le système croisé d'événements de navigateur. Les événements sont joints en tant que propriétés des composants et peuvent déclencher des méthodes ou des classes.

La gestion des événements pour les éléments React est très similaire à celle des éléments du DOM.

Il y a tout de même quelques différences de syntaxe :

Les événements de React sont nommés en camelCase plutôt qu'en minuscules.

En JSX on passe une fonction comme gestionnaire d'événements plutôt qu'une chaîne de caractères.

React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Click

Nous allons maintenant "surveiller" les événements et déclencher des actions avec l'aide d'une fonction

```
const ButtonMessage = props => {
  const traiterClick = params => {
    console.info("Click!");
  };
  return <button onClick={traiterClick}>Click me!</button>;
};
ReactDOM.render(<ButtonMessage />, document.querySelector(".eve"));
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Click

Nous allons maintenant "surveiller" les événements et déclencher des actions

Si vous ne souhaitez pas utiliser bind, vous avez une alternatives possibles. En utilisant des classes,

```
const App = () => {
  const handleClick = event => {
    console.info(event);
  };
  return <button onClick={handleClick}>GO</button>;
};
ReactDOM.render(<App />, document.querySelector(".eve"));
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Click

Nous allons maintenant "surveiller" les événements et déclencher des actions

Si nous le faisons avec les Hook,

```
const ButtonMessage = props => {
  const traiterClick = params => {
    console.info("Click!");
  };
  return <button onClick={traiterClick}>Click me!</button>;
};
const GreeterMessage = ({whom}) => {
  const traiterClick = params => {
    console.info(` Bonjour, ${whom}! `);
  };
  return <button onClick={traiterClick}>Click here!</button>;    };
ReactDOM.render(<ButtonMessage />, document.querySelector(".eve"));
ReactDOM.render(<GreeterMessage whom="Stephane" />,
document.querySelector(".eve2"));
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Resize

L'événement resize de window est un comportement que l'on peut manipuler avec React.

```
class WindowWidth extends React.Component {  
  constructor() {  
    super();  
    this.state = {  
      height: window.innerHeight,  
      width: window.innerWidth  
    };  
    this.updateDimensions = this.updateDimensions.bind(this);  
  }  
  componentDidMount() {  
    console.info(this.state.height);  
    // Additionally I could have just used an arrow function for the binding  
    `this` to the component...  
    window.addEventListener("resize", this.updateDimensions);  
  }  
  updateDimensions() {  
    this.setState({  
      height: window.innerHeight,  
      width: window.innerWidth  
    });  
  } ...
```



React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Resize

L'événement resize de window est un comportement que l'on peut manipuler avec React.

... suite

```
...
render() {
  return (
    <h3>
      Window width: {this.state.width} and height: {this.state.height}
    </h3>
  );
}
componentWillUnmount() {
  window.removeEventListener("resize", this.updateDimensions);
}
}

ReactDOM.render(<WindowWidth />, document.querySelector(".eve"));
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des événements

Resize

L'événement resize de window est un comportement que l'on peut manipuler avec React.

... suite

```
...
render() {
  return (
    <h3>
      Window width: {this.state.width} and height: {this.state.height}
    </h3>
  );
}
componentWillUnmount() {
  window.removeEventListener("resize", this.updateDimensions);
}
}

ReactDOM.render(<WindowWidth />, document.querySelector(".eve"));
```

Interactivité des composants

Gestion des données statiques

DataFlow

Affichez souvent un modèle de données JSON à un utilisateur, entraîne logiquement la séparation de votre interface utilisateur en composants.

Découpez-le simplement en composants qui représentent exactement un élément de votre modèle de données

FilterableProductTable : contient l'intégralité de l'exemple

SearchBar : reçoit toutes les entrées de l'utilisateur

ProductTable : affiche et filtre la collecte de données en fonction des entrées de l'utilisateur

ProductCategoryRow : affiche un titre pour chaque catégorie

ProductRow : affiche une ligne pour chaque produit

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette doivent apparaître en tant qu'enfant dans la hiérarchie :
FilterableProductTable

- SearchBar
- ProductTable
 - ProductCategoryRow
 - ProductRow

A la fin de cette étape, nous aurons une bibliothèque de composants réutilisable pour afficher un modèle de données
La méthode Render() pour afficher des données statiques

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
class ProductCategoryRow extends React.Component {  
  render() {  
    const category = this.props.category;  
    return (  
      <tr>  
        <th colSpan="2">  
          {category}  
        </th>  
      </tr>  
    );  
  }  
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
class ProductRow extends React.Component {  
  render() {  
    const product = this.props.product;  
    const name = product.stocked ?  
      product.name :  
      <span style={{color: 'red'}}>  
        {product.name}  
      </span>;  
  
    return (  
      <tr>  
        <td>{name}</td>  
        <td>{product.price}</td>  
      </tr>  
    );  
  }  
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
class ProductTable extends React.Component {  
  render() {  
    const filterText = this.props.filterText;  
    const inStockOnly = this.props.inStockOnly;  
  
    const rows = [];  
    let lastCategory = null;  
  
    this.props.products.forEach((product) => {  
      if (product.name.indexOf(filterText) === -1) {  
        return;  
      }  
      if (inStockOnly && !product.stocked) {  
        return;  
      }  
      if (product.category !== lastCategory) {  
        rows.push(  
          <ProductCategoryRow  
            category={product.category}  
            key={product.category} />  
        );  
      }  
    });  
  }  
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
rows.push(  
  <ProductRow  
    product={product}  
    key={product.name}  
  />  
);  
lastCategory = product.category;  
});  
  
return (  
  <table>  
    <thead>  
      <tr>  
        <th>Name</th>  
        <th>Price</th>  
      </tr>  
    </thead>  
    <tbody>{rows}</tbody>  
  </table>  
);  
}  
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
class SearchBar extends React.Component {  
  constructor(props) {  
    super(props);  
    this.handleFilterTextChange = this.handleFilterTextChange.bind(this);  
    this.handleInStockChange = this.handleInStockChange.bind(this);  
  }  
  
  handleFilterTextChange(e) {  
    this.props.onFilterTextChange(e.target.value);  
  }  
  
  handleInStockChange(e) {  
    this.props.onInStockChange(e.target.checked);  
  }  
}
```



React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
render() {
  return (
    <form>
      <input
        type="text"
        placeholder="Search..."
        value={this.props.filterText}
        onChange={this.handleFilterTextChange}
      />
      <p>
        <input
          type="checkbox"
          checked={this.props.inStockOnly}
          onChange={this.handleInStockChange}
        />
        {' '}
        Only show products in stock
      </p>
    </form>
  );
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
class FilterableProductTable extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      filterText: "",  
      inStockOnly: false  
    };  
  
    this.handleFilterTextChange = this.handleFilterTextChange.bind(this);  
    this.handleInStockChange = this.handleInStockChange.bind(this);  
  }  
  
  handleFilterTextChange(filterText) {  
    this.setState({  
      filterText: filterText  
    });  
  }  
  
  handleInStockChange(inStockOnly) {  
    this.setState({  
      inStockOnly: inStockOnly  
    })  
  }  
}
```

React.JS

un framework JavaScript

Interactivité des composants

Gestion des données statiques

DataFlow

Les composants apparaissant dans un autre composant de la maquette

```
render() {
  return (
    <div>
      <SearchBar
        filterText={this.state.filterText}
        inStockOnly={this.state.inStockOnly}
        onFilterTextChange={this.handleFilterTextChange}
        onInStockChange={this.handleInStockChange}>
      />
      <ProductTable
        products={this.props.products}
        filterText={this.state.filterText}
        inStockOnly={this.state.inStockOnly}>
      />
    </div>
  );
}
```



HOC

Écrire des composants réutilisables est l'un des objectifs de tout développeur React.

Que ce soit pour créer une boîte à outils de composants stylés, pour mutualiser des comportements communs

un HOC est une fonction qui prend en paramètre une définition de composant (classe ou fonction), et renvoie une nouvelle définition de composant, qui ajoute du comportement à la première



React.JS

un framework JavaScript

Interactivité des composants

Higher Order Components

HOC high-order components

HOC

Écrire des composants réutilisables est l'un des objectifs de tout développeur React.

Que ce soit pour créer une boîte à outils de composants stylés, pour mutualiser des comportements communs

un HOC est une fonction qui prend en paramètre une définition de composant (classe ou fonction), et renvoie une nouvelle définition de composant, qui ajoute du comportement à la première

```
const addBorder = borderWidth => Component => props => (
  <div style={{ borderColor: "black", borderStyle: "solid", borderWidth }}>
    <Component {...props} />
  </div>
);
const MyText = <p>Hello!</p>;
const MyTextWithBorder = addBorder(5)(MyText);
```



HOC

Vous obtenez un composant MyTextWithBorder qui affiche le texte « Hello » avec une bordure

addBorder est ce que l'on appelle un high-order component.

```
const addBorder = borderWidth => Component => props => (
  <div style={{ borderColor: "black", borderStyle: "solid", borderWidth }}>
    <Component {...props} />
  </div>
);
const MyText = <p>Hello!</p>;
const MyTextWithBorder = addBorder(5)(MyText);
```





React.JS

un framework JavaScript

Interactivité des composants

HOC high-order components

HOC

Comme exemple, nous allons utiliser le concept d'HOC pour créer un champ de saisie de numéro de téléphone, qui :

n'acceptera que les chiffres, parenthèses, tirets et espaces en entrée (à la frappe) ;

mettra en forme le numéro de téléphone

lorsque le focus sera perdu par le champ (événement *blur*).
(Seuls les numéros de téléphone Nord-Américains seront pris en compte : « (514) 555-0199 ».)



React.JS

un framework JavaScript

Interactivité des composants

Higher Order Components

HOC high-order components

HOC

notre champs sera contrôlé, c'est-à-dire que nous utiliserons les propriétés value et onChange pour savoir quel texte afficher et comment le mettre à jour.

Nous souhaitons également que la valeur ne contienne que les chiffres du numéro de téléphone (« 5145550199 »), sans se soucier de la mise en forme, et donc que le onChange soit appelé avec les chiffres uniquement (dans event.target.value).

```
const PhoneNumberInput = formatPhoneNumber(  
  forbidNonPhoneNumberCharacters(props => <input {...props} />),  
);
```



React.JS

un framework JavaScript

Interactivité des composants

HOC high-order components

HOC

fonction de Recompose que nous utiliserons : compose.
Elle effectue la composition de plusieurs HOC pour
les fusionner en un seul

```
const PhoneNumberInput = compose(  
  formatPhoneNumber, forbidNonPhoneNumberCharacters,  
)(props => <input {...props} />);
```



React.JS

un framework JavaScript

Interactivité des composants

Higher Order Components

HOC high-order components

HOC

Pour rendre nos HOC réutilisable, rendons-les plus génériques :

```
// Ne garde que les chiffres, espaces, tirets et parenthèses
const forbiddenCharactersInPhoneNumber = /^[^\d\s\-\()]/g;
// '5145551234' => '(514) 555-1234'
const formatPhoneNumber = value =>
  value.replace(/^(\\d{3})(\\d{3})(\\d{4})$/,"($1) $2-$3");
// '(514) 555-1234' => '5145551234'
const parsePhoneNumber = formattedPhoneNumber =>
  formattedPhoneNumber.replace(/[^\\d]/g, "").slice(0, 10);
const PhoneNumberInput = compose(
  formatInputValue({
    formatValue: formatPhoneNumber,
    parseValue: parsePhoneNumber,
  }),
  forbidCharacters(forbiddenCharactersInPhoneNumber),
)(props => <input {...props} />);
```



React.JS

un framework JavaScript

Interactivité des composants

HOC high-order components

HOC

Lorsque la valeur de l'input est changée (évènement **onChange**), on intercepte cet évènement pour supprimer tout caractère interdit de la valeur, puis on appelle la propriété **onChange** parente avec la valeur propre.

Nous utiliserons ici la fonction **withHandlers** pour ajouter des nouveaux handlers d'évènement comme propriétés du composant encapsulé.

Nous avons accès aux propriétés de notre composant (ici nous utiliserons **onChange**) pour créer notre nouveau handler :



React.JS

un framework JavaScript

Interactivité des composants

HOC high-order components

HOC

```
const forbidCharacters = forbiddenCharsRegexp =>
withHandlers({
  onChange: props => event => {
    // N'oublions pas que `onChange` n'est pas une propriété requise
    // (même si rien ne se produira si elle est absente).
    if (props.onChange) {
      const value = event.target.value;
      const cleanValue = value.replace(forbiddenCharsRegexp, "");
      // On ne modifie pas l'évènement original, mais on le clone
      // en y redéfinissant event.target.value avec la valeur propre.
      const newEvent = {
        ...event,
        target: { ...event.target, value: cleanValue },
      };
      // On réémet notre évènement au `onChange` parent.
      props.onChange(newEvent);
    }
  },
});
```





React.JS

un framework JavaScript

Interactivité des composants

Higher Order Components

HOC high-order components

HOC

Le second pour la mise en forme

```
const formatInputValue = ({ formatValue, parseValue }) =>
compose(
  useState("inputValue", "setInputValue", props =>
    formatValue(props.value)),
  withHandlers({
    onChange: props => event => {
      props.setInputValue(event.target.value);
    },
    onBlur: props => event => {
      const parsedValue = parseValue(props.inputValue);
      const formattedValue = formatValue(parsedValue);
      props.setInputValue(formattedValue);
      const newEvent = {
        ...event,
        target: { ...event.target, value: parsedValue },
      };
      if (props.onChange) {
        props.onChange(newEvent);
      }
      if (props.onBlur) {
        props.onBlur(newEvent);
      }
    },
    mapProps(props => ({
      ...omit(props, ["inputValue", "setInputValue"]),
      value: props.inputValue,
    })), );
)
```



HOC

L'exemple ci-dessus prend un composant et inverse le contenu qu'il contient. reverse est un HOC, qui prend un élément (name dans l'exemple), trouve le contenu à l'intérieur de cet élément, l'inverse et crache un élément avec un contenu inversé.

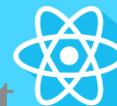
A finir

```
const reverse = (PassedComponent) =>
  ({ children, ...props } ) =>
    <PassedComponent {...props}>
      {children.split("").reverse().join("")}
    </PassedComponent>
```

```
const name = (props) => <span>{props.children}</span>
const reversedName = reverse(name)
<reversedName>Hello</reversedName>

//=> <span>olleH</span>
```





Interactivité des composants

Gestion des formulaires

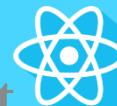
Les formulaires

Nous allons voir ensemble comment faire pour traiter plus facilement et de manière plus optimisé plusieurs champs afin de l'enregistrer dans notre state.

Nous allons prendre pour exemple un formulaire de connexion qui prendra un compte un email et un mot de passe.

Pour commencer, il faut créer notre composant ReactJS

```
class Login extends Component {  
  render () {  
    return (  
      <form>  
        <label>Email</label>  
        <input type="text" name="email" />  
        <label>Mot de passe</label>  
        <input type="password" name="password" />  
      </form> );  
    }  
}
```



Interactivité des composants

Gestion des formulaires

Les formulaires

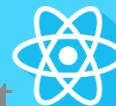
Quand la valeur d'un champ va changer (onChange) la fonction handleForm va être appelée.

Elle va prendre en paramètre les informations du champ courant.

On modifie le state avec en clé la propriété « name » des champs (email ou password dans notre cas) et va enregistrer le contenu du champs comme valeur dans le state.

```
render () {
  return (
    <form>
      <label>Email</label>
      <input type="email" name="email" onChange={this.handleForm} />
      <label>Mot de passe</label>
      <input type="password" name="password"
        onChange={this.handleForm} />
    </fom>
  );
}
ReactDOM.render(<Login />, document.querySelector(".formulaire"));
```





Interactivité des composants

Gestion des formulaires

Les formulaires

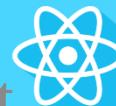
Quand la valeur d'un champ va changer (onChange) la fonction handleForm va être appelée.

Elle va prendre en paramètre les informations du champ courant.

On modifie le state avec en clé la propriété « name » des champs (email ou password dans notre cas) et va enregistrer le contenu du champs comme valeur dans le state.

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

    this.state = {
      loading: false,
      success: false,
      error: false
    }
  }
  // Actions
  handleSubmit(ev) {
    ev.preventDefault();
    this.setState({loading: true})
  }
}
```



Interactivité des composants

Gestion des formulaires

Les formulaires

En HTML, les éléments de formulaire tels que <input>, <textarea> et <select> conservent généralement leur propre état et le mettent à jour en fonction des entrées de l'utilisateur.

Dans React, l'état mutable est généralement conservé dans la propriété state des composants et uniquement mis à jour avec setState().

Puisque la value attribut est défini sur notre élément de formulaire, la valeur affichée le sera toujours this.state.value, ce qui fera de l'état React la source de la vérité.

Puisqu'il est handleChange exécuté sur chaque frappe pour mettre à jour l'état de réaction, la valeur affichée sera mise à jour au fur et à mesure que l'utilisateur tape.

Avec un composant contrôlé, chaque mutation d'état aura une fonction de gestionnaire associée.

Cela facilite la modification ou la validation de la saisie de l'utilisateur



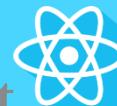
Interactivité des composants

Gestion des formulaires

Les formulaires

En HTML, les éléments de formulaire tels

```
class NameForm extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {value: ''};  
  
    this.handleChange = this.handleChange.bind(this);  
    this.handleSubmit = this.handleSubmit.bind(this);  
  }  
  
  handleChange(event) {  
    this.setState({value: event.target.value});  
  }  
  
  handleSubmit(event) {  
    alert('A name was submitted: ' + this.state.value);  
    event.preventDefault();  
  }  
  ....
```



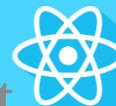
Interactivité des composants

Gestion des formulaires

Les formulaires

En HTML, les éléments de formulaire tels

```
....  
render() {  
    return (  
        <form onSubmit={this.handleSubmit}>  
            <label>  
                Name:  
                <input type="text" value={this.state.value}  
onChange={this.handleChange} />  
            </label>  
            <input type="submit" value="Submit" />  
        </form>  
    );  
}  
ReactDOM.render(<NameForm />, document.querySelector(".formulaire"));
```



Interactivité des composants

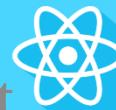
Gestion des formulaires

Les formulaires

Champs select

```
const VinForm = () => {
  const [value, setValue] = React.useState("cornas");
  const handleChange = event => {
    setValue(event.target.value);
  };
  const handleSubmit = event => {
    alert("Région viticole : "+value);
    event.preventDefault();
  };
  return (
    <form onSubmit={handleSubmit}>
      <label>
        Choisir une région :&nbsp;
        <select value={value} onChange={handleChange}>
          <option value="bordeaux">Bordeaux</option>
          <option value="bourgogne">Bourgogne</option>
          <option value="alsace">Alsace</option>
          <option value="cornas">Cornas</option>
        </select>
      </label>
      &nbsp;
      <input type="submit" value="Submit" />
    </form>
  );
}
```





Interactivité des composants

Gestion des formulaires

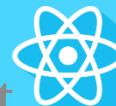
Les formulaires

Champs select

```
</form>
);
};

ReactDOM.render(<VinForm />, document.querySelector(".formulaire"));
```





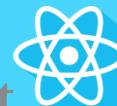
Interactivité des composants

Gestion des formulaires

Les formulaires

Champs texarea

```
const ZoneT = () => {
  const [text, setText] = React.useState(
    "Auto BR\nAuto Link: https://sbrunet.fr/react/"
  );
  const changeText = event => {
    setText(event.target.value);
  };
  const editText = text => {
    const regExp = /(https?:\/\/\S+|\n)/;
    const regExpBr = /\n/;
    const regExpLink = /https?:\/\/\S+/;
    return text.split(regExp).map(function(line, i) {
      return line.match(regExpBr) ? (<br key={i} />) : line.match(regExpLink) ? (
        <a target="_blank" href={line} key={i}>{line}</a>) : (line);
    });
  };
  return (
    <>
      <textarea onChange={changeText} defaultValue={text} />
      <p>{editText(text)}</p>
    </>
  );
};
ReactDOM.render(<ZoneT />, document.querySelector(".formulaire"));
```



Interactivité des composants

Gestion des formulaires

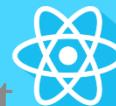
Les formulaires

Champs input et datalist avec un hook

...

```
const Star = () => {
  const [items, setItems] = React.useState([]);
  React.useEffect(() => {
    fetch("https://swapi.co/api/people/?format=json")
      .then(response => response.json())
      .then(json => setItems(json.results));
  }, []);
  return (
    <div>
      <input type="text" list="sw" />
      <datalist id="sw">
        {items.map(item => (
          <Person key={item.name} person={item} />
        ))}
      </datalist>
    </div>
  );
}
const Person = props => <option value={props.person.name}>
/>;ReactDOM.render(<Star />, document.querySelector(".formulaire"));
```





Interactivité des composants

Gestion des formulaires

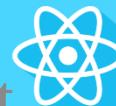
Les formulaires

Champs input et datalist avec un hook

...

```
const Star = () => {
  const [items, setItems] = React.useState([]);
  React.useEffect(() => {
    fetch("https://swapi.co/api/people/?format=json")
      .then(response => response.json())
      .then(json => setItems(json.results));
  }, []);
  return (
    <div>
      <input type="text" list="sw" />
      <datalist id="sw">
        {items.map(item => (
          <Person key={item.name} person={item} />
        ))}
      </datalist>
    </div>
  );
}
const Person = props => <option value={props.person.name}>
/>;ReactDOM.render(<Star />, document.querySelector(".formulaire"));
```





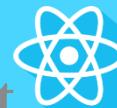
Interactivité des composants

Gestion des formulaires

Evènement Ref pour accéder au DOM

```
function Reffonc() {
  const ref = React.createRef();
  const focus = () => ref.current.focus();
  return (
    <React.Fragment>
      <p>Functional component</p>
      <input type="text" ref={ref} />
      <button onClick={focus}>Click me</button>
    </React.Fragment>
  );
}
ReactDOM.render(<Reffonc />, document.querySelector(".formulaire3"));
```





Interactivité des composants

Gestion des formulaires

Lazy load

React.lazy prend une fonction qui doit appeler un import() dynamique.

Ça doit renvoyer une Promise qui s'accomplit avec un module dont l'export par défaut contient un composant React.

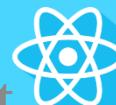
Remarque : React.lazy et Suspense ne sont pas encore disponibles pour le rendu côté serveur.

```
const { lazy, Suspense } = React;

const Lazy = lazy(() => new Promise(resolve => {
  setTimeout(() => {
    resolve({ default: () => <Resource /> });
  }, 4000);
});

const Resource = () => (
  <div className="box">
    <h3>React Lazy</h3>
    <p>Block chargé après 4 seconds, utilise React Lazy et Suspense</p>
  </div>
)

const App = () => {
  return (
    //Suspense, permet d'afficher un contenu de repli
    <Suspense fallback={ <div><br />Loading...</div> }>
      <Lazy/>
    </Suspense>
  )
}
ReactDOM.render(<App />, document.querySelector(".ajax"));
```



Interactivité des composants

Gestion des formulaires

Locale storage

A l'aide de local storage HTML5

```
const setEntriesToStorage = items =>
window.localStorage.setItem('journalEntries', JSON.stringify(items));
useEffect(() => {
  const entriesFromStorage = getEntriesFromStorage();
  if(entriesFromStorage) {
    setEntries(entriesFromStorage);
  }
}, []);
const storeEntry = (entry) => {
  const newEntries = [entry, ...entries];
  setEntries(newEntries);
  setEntriesToStorage(newEntries);
}
```

React.JS

un framework JavaScript

- Rappels des composants des RIA
- Développer avec ReactJS
- Interactivité des composants
- Application monopage avec ReactJS Redux
- Application isomorphique
- Introduction à React Native



- Flux/Redux : présentation. Propagation de données.
- Comparaison des architectures.
- Création de vues et contrôleurs dans Flux.
- Rôle du "Dispatcheur" dans Flux pour les actions.
- Les "Stores", gestionnaire d'états logique dans Flux.
- Définition du Functional Programming.
- Approche avec Redux. Le "Reducer".

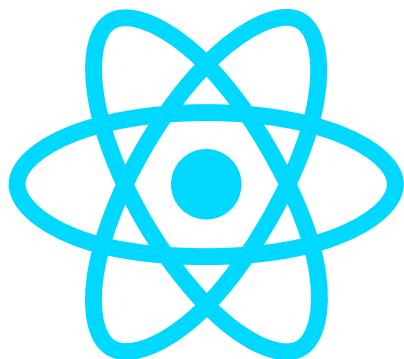
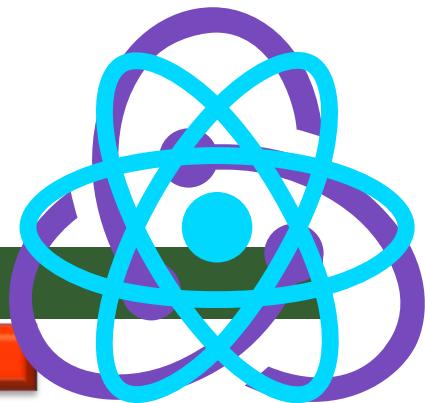


React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Create React App

Create React App est un environnement confortable pour démarrer une nouvelle application web monopage en React.

Il configure votre environnement de développement de façon à vous permettre d'utiliser les dernières fonctionnalités de JavaScript, propose une expérience développeur agréable et optimise votre application pour la production.

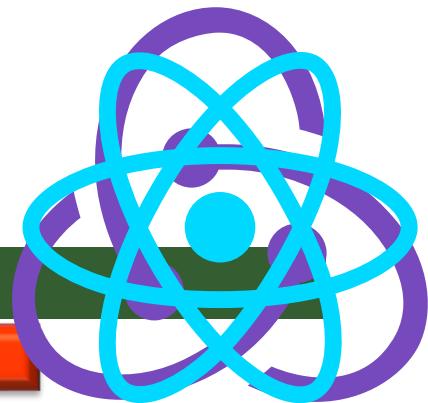
Nous aurons besoin de Node ≥ 6 et de npm ≥ 5.2 sur votre machine.

React.JS

un framework JavaScript

Application monopage

Node serveur js



Node JS

Node.js agit dans un autre contexte : il vous permet d'utiliser JavaScript côté serveur, en dehors du navigateur.

Avant de pouvoir exécuter votre JavaScript sur le serveur, le code a besoin d'être interprété.

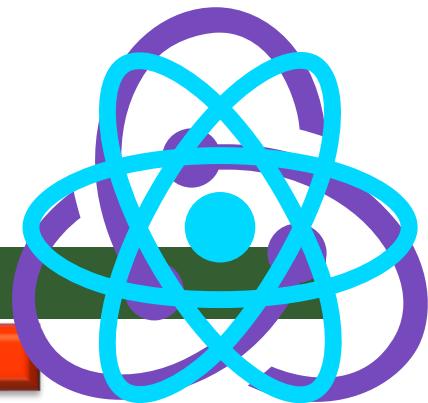
C'est précisément ce que fait Node.js en utilisant le moteur V8 de Google, celui qui est utilisé par le navigateur Chrome.

Ainsi, Node.js comprend deux éléments distincts : un contexte d'exécution et une bibliothèque.



React.JS

un framework JavaScript



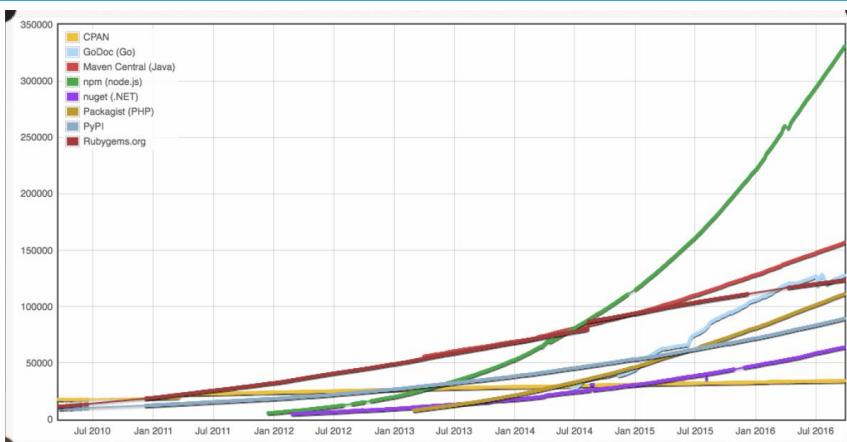
Application monopage

Node serveur js

Node JS

NODE.JS PARTOUT • CHRISTOPHE PORTENEUVE @ PARIS WEB 2016

NODE.JS EST LA PLATE-FORME OPEN-SOURCE À LA PLUS FORTE CROISSANCE, AU NIVEAU MONDIAL.

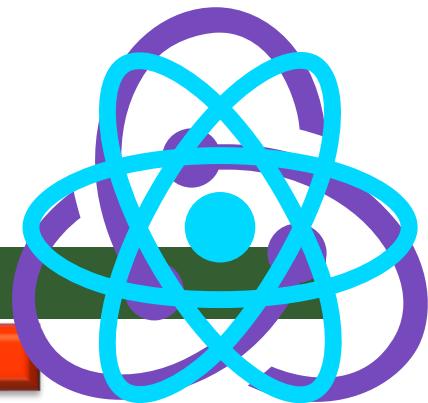


React.JS

un framework JavaScript

Application monopage

Node serveur js



Node JS

La deuxième techno la plus demandée après Android

La quasi totalité du Fortune 500 a migré vers Node.js (généralement depuis Java EE), avec des résultats époustouflants.

Netflix : bootup –98%, instances EC2 –75%, latence réduite !

PayPal : RPS x 2, temps de réponse –35%

Groupon : temps de chargement de page –50%

Uber, LinkedIn, Walmart, NASA, Yahoo!, eBay, Medium, Trello, Storify, SitePen, JSBin, Yammer, Zendesk...

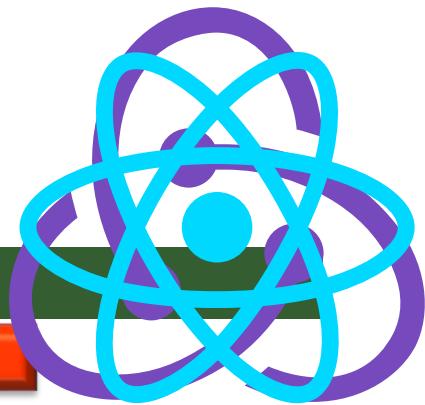


React.JS

un framework JavaScript

Application monopage

Node serveur js



Node JS

Commande

C:\nodejs> console.info(2+2)

Node expression interprétée

C:\nodejs> npm install module-xx

installation du module "module-xx"

C:\nodejs> npm install --global module-xx

installation du module "module-xx" global réutilisable

C:\nodejs> npm uninstall module-xx

desinstallation du module "module-xx"

C:\nodejs> npm module-xx --version

La version du module

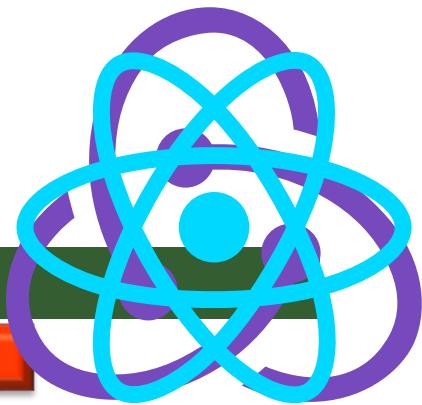
C:\nodejs> npm run

démarrer l'application js



React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Create React App

```
C:\ Invité de commandes
M- uninstall
M- unstar

C:\nodejs>npm install -g create-react-app
C:\Users\Stéphane\AppData\Roaming\npm\create-react-app -> C:\Users\Stéphane\AppData\Roaming\npm\node_modules\create-react-app\index.js
+ create-react-app@2.1.8
added 63 packages from 20 contributors in 25.149s

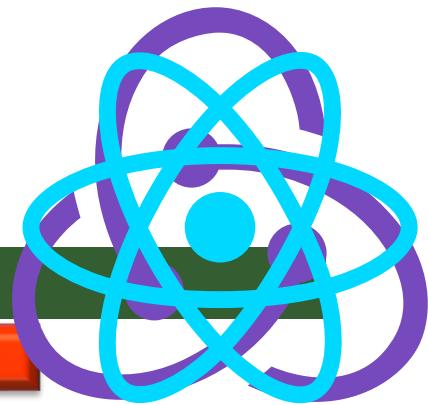
C:\nodejs>
```

C:\nodejs> npm install -g create-react-app

C:\nodejs> create-react-app 1-walkman

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Create React App

```
• npm start
  Starts the development server.

  • npm run build
    Bundles the app into static files for production.

  • npm test
    Starts the test runner.

  • npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

  • I suggest that you begin by typing:

    cd reactapp
    npm start

  Happy hacking!

  C:\nodejs>create-react-app -v
  Please specify the project directory:
  create-react-app <project-directory>

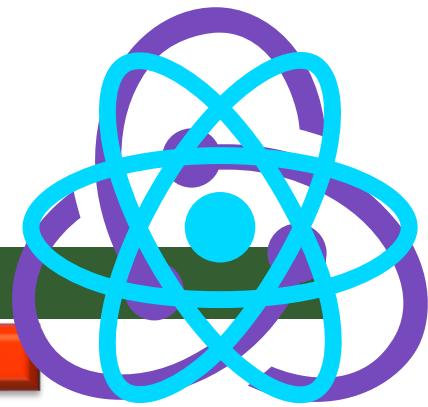
  Or example:
  create-react-app my-react-app

  Run create-react-app --help to see all options.
```

C:\nodejs> create-react-app -v

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Create React App

```
npm
Starts the test runner.

npm run eject
Removes this tool and copies build dependencies, configuration files
and scripts into the app directory. If you do this, you can't go back!

suggest that you begin by typing:

cd reactapp
npm start

appy hacking!

:\nodejs>create-react-app -v
Please specify the project directory:
create-react-app <project-directory>

or example:
create-react-app my-react-app

In create-react-app --help to see all options.

:\nodejs>cd reactapp
:\nodejs\reactapp>npm start

reactapp@0.1.0 start C:\nodejs\reactapp
react-scripts start
```

C:\nodejs> create-react-app reactapp

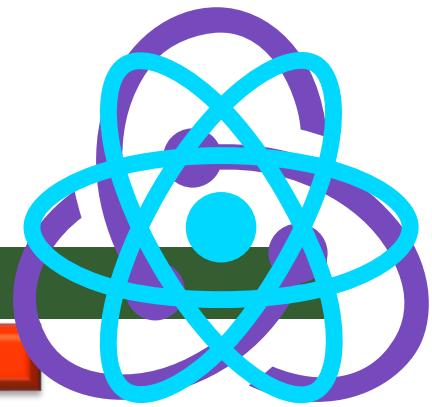
C:\nodejs> cd reactapp

C:\nodejs> reactapp> npm start



React.JS

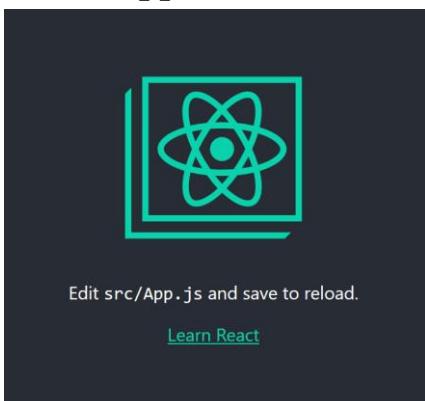
un framework JavaScript



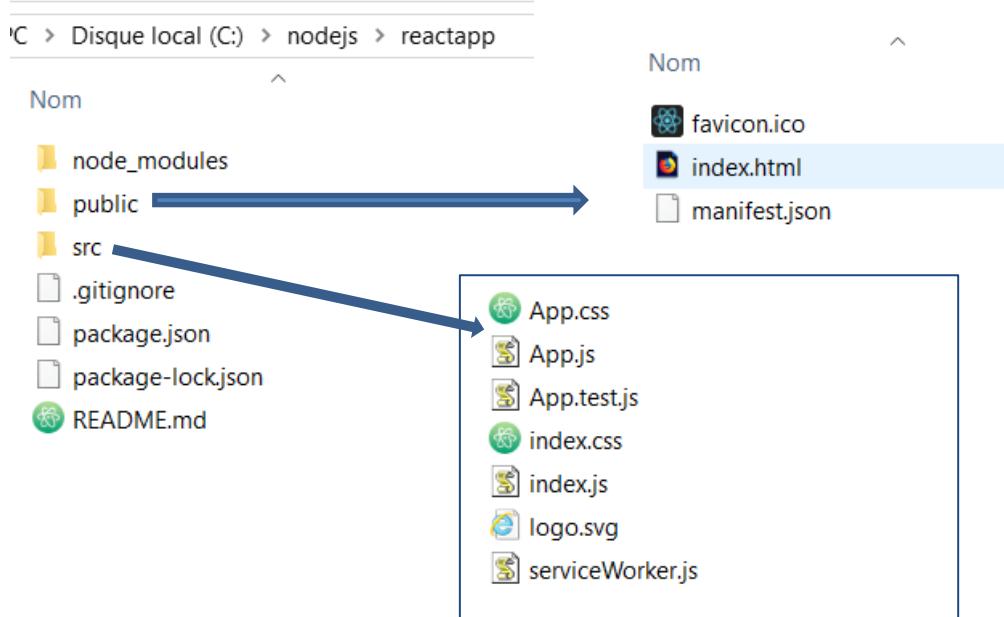
Application monopage

Javascript coté serveur

Create React App

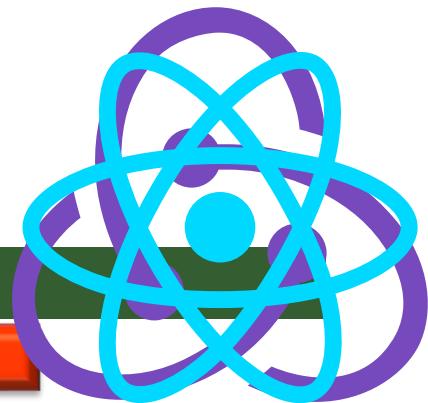


<http://localhost:3000/>



React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Create React App

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

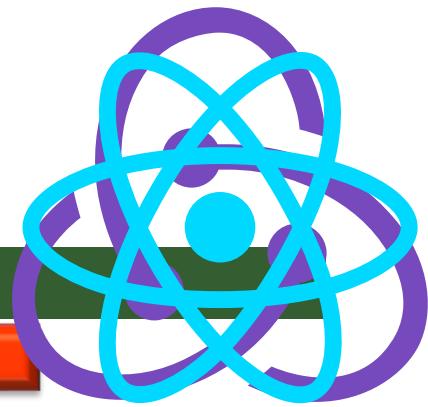
ReactDOM.render(<App />, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

src/index.js

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Create React App

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

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

// If you want your app to work offline and load faster, you can change
// unregister() to register() below. Note this comes with some pitfalls.
// Learn more about service workers: https://bit.ly/CRA-PWA
serviceWorker.unregister();
```

src/app.test.js

permet de faire des tests sur le composant app.js

Pour effectuer les tests

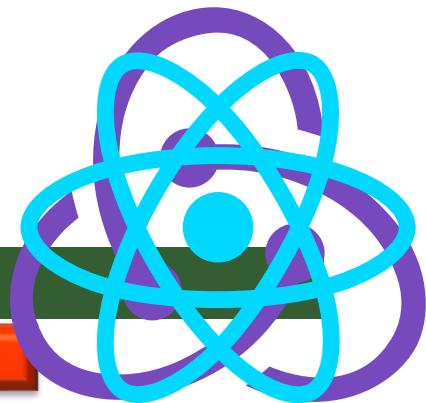
C:\nodejs> reactapp> npm test

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



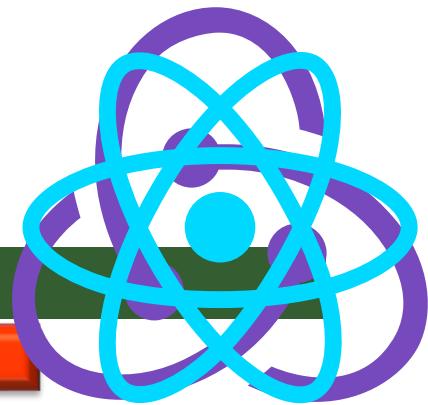
Create React App

```
class App extends Component {  
  render(){  
    return(  
      <div className="App">  
        <div className="App-header">  
          <img src={logo} className="App-logo" Alt="app-logo" />  
          <h2> Bienvenu </h2>  
        </div>  
        <p className="App-intro"> Formation React </p>  
      </div>  
    );  
  }  
}  
export default App;
```

C:\nodejs> reactapp> npm start

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

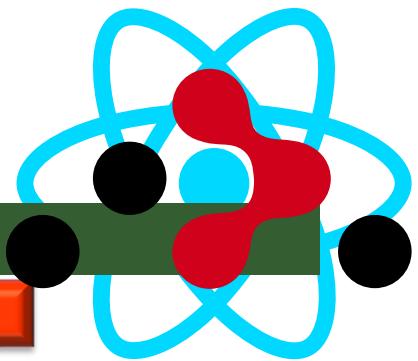
Create React App

- Structure des fichiers pour créer ce composant on a deux options :
 - * Un Composant React complet (une class qui étends React.Component).
 - * Une fonction pure.

```
1 import React from 'react';
2 import { render } from 'react-dom';
3
4 import Counter from './Counter'
5
6 render( <Counter />, document.querySelector('#root'))
7
```

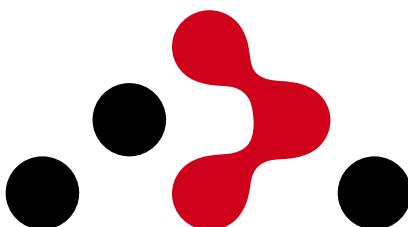
public
src-struct
Counter.jsx
index.jsx

C:\nodejs> reactapp> npm start



Application monopage

Router



Utilisation du module react-router

React Router est une librairie qui affiche vos composants en fonction de l'URL de votre navigateur indispensable si vous voulez faire une SPA

L'idée de Router (Routeur) est tellement utile quand vous travaillez avec React, une bibliothèque Javascript pour programmer des applications d'une seule page **SPA** (Single Page Application).

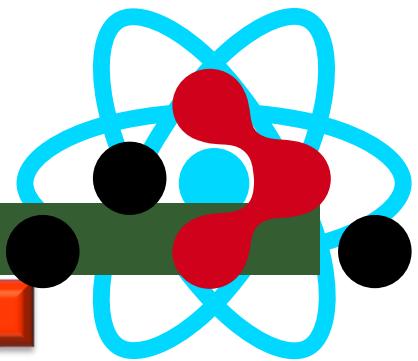
Afin de développer une application React il vous faut écrire plusieurs Component mais il a besoin d'un seul fichier afin de servir des utilisateur, c'est index.html.

React.JS

un framework JavaScript

Application monopage

Javascript côté serveur



Utilisation du module react-router

React Router

C:\nodejs> 2-router> npm install react-router-dom

Il faut installer le module

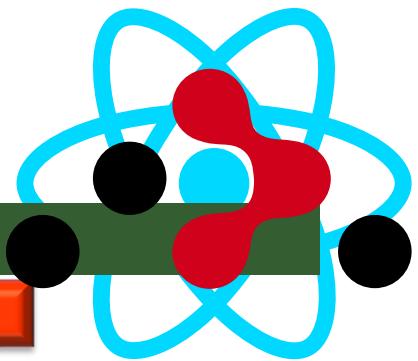
Le **React Router** vous permet de définir des **URL** dynamiques
Et de sélectionner un **Component** approprié pour **render** (afficher)
sur le navigateur d'utilisateur en correspondance à chaque **URL**.

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

Le **React Router** vous fournit deux composants tels que **<BrowserRouter>** & **<HashRouter>**.

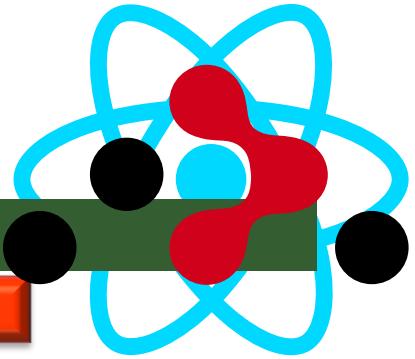
Ces deux composants sont différents dans le type de **URL** qu'ils créent et synchronisent avec

React.JS

un framework JavaScript

Application monopage

Javascript côté serveur



Utilisation du module react-router

React Router

<BrowserRouter> est utilisé plus couramment, il utilise le **History API** incluse dans **HTML5** pour surveiller l'historique de votre routeur tandis que <HashRouter> utilise le **hash** de l' **URL (*window.location.hash*)** pour tout mémoriser.

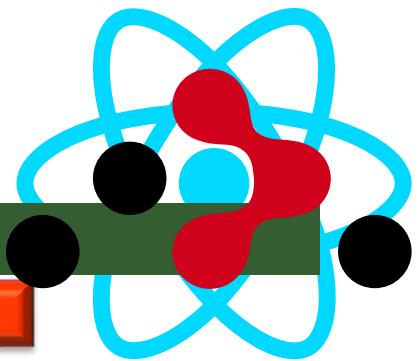
Si vous avez l'intention de soutenir des anciens navigateurs, vous devez être scellé contre le <HashRouter> ou créer une application **React** à l'aide du routeur côté client. <HashRouter> est un choix raisonnable.

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

Le composant **<Route>** définit une relation (mapping) entre une **URL** et un **Component**.

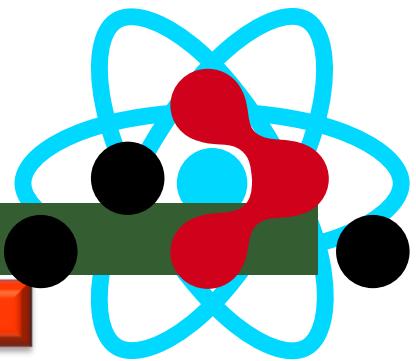
Cela signifie que lorsque l'utilisateur visite une **URL** sur le navigateur, un **Component** correspondant doit être rendu sur l'interface.

L'attribut `exact` est utilisé dans le **<Route>** afin de dire que ce **<Route>** ne fonctionne que si la URL sur le navigateur correspond absolument à la valeur de son attribut `path`.

```
<BrowserRouter>
  <Route exact path="/" component={Home}/>
  <Route path="/apropos" component={apropos}/>
  <Route path="/sujet" component={sujet}/>
</BrowserRouter>
```

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

React Router

```
cd react-router-basic-app  
npm install --save react-router-dom
```

```
cd react-router-app  
npm start
```

Happy hacking!

```
C:\nodejs>cd react-router-app  
  
C:\nodejs\react-router-app>npm install --save react-router-dom  
npm [WARN] deprecated core-js@1.2.7: core-js@<2.6.5 is no longer maintained. Please, upgrade to c  
ctual version of core-js@2.  
npm [WARN] ts-pnp@1.1.2 requires a peer of typescript@* but none is installed. You must install p  
. .  
npm [WARN] optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):  
npm [WARN] notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {  
"current": {"os": "win32", "arch": "x64"}  
npm [WARN] optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.8 (node_modules\chokidar\node_modu  
npm [WARN] notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.8: wanted {  
"current": {"os": "win32", "arch": "x64"}  
  
+ react-router-dom@5.0.0  
added 19 packages from 89 contributors and audited 36313 packages in 132.645s  
found 68 vulnerabilities (63 low, 5 high)  
  run `npm audit fix` to fix them, or `npm audit` for details  
  
C:\nodejs\react-router-app>
```

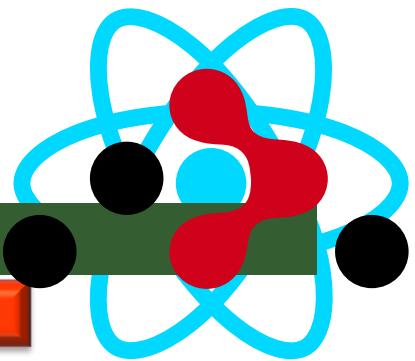


React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

Le fichier package.json a été mis à jour

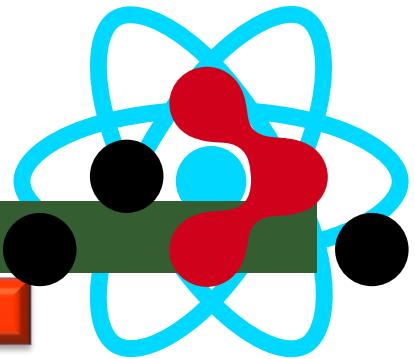
```
{  
  "name": "react-router-app",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "react": "^16.8.6",  
    "react-dom": "^16.8.6",  
    "react-router-dom": "^5.0.0",  
    "react-scripts": "2.1.8"  
  },
```

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

Le fichier package.json a été mis à jour

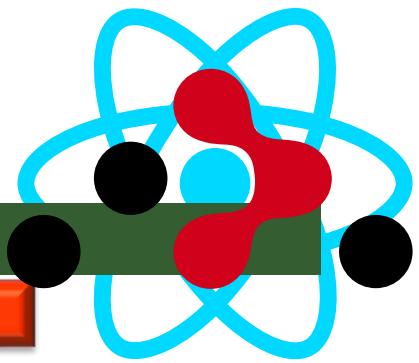
npm start

Nous allons lancer le serveur

```
C:\nodejs\react-router-app>npm start
> react-router-app@0.1.0 start C:\nodejs\react-router-app
> react-scripts start
```

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

React Router

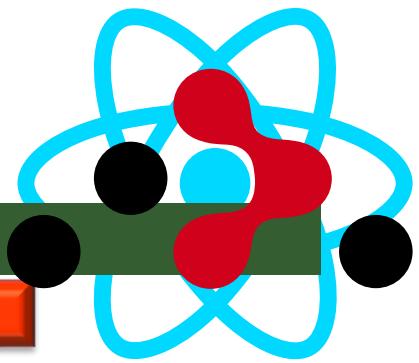
Le fichier App.css

```
.main-route-place {  
  border: 1px solid #bb8fce;  
  margin: 3px;  
  padding: 5px;  
}
```

```
.secondary-route-place {  
  border: 1px solid #a2d9ce;  
  margin: 5px;  
  padding: 5px;  
}
```

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

React Router

Le fichier App.js

```
import React from "react";
import { BrowserRouter, Route, Link, Switch } from "react-router-dom";
import './App.css';
function App(props){
  return(
    <Switch>
      <Route path="/" render={()=><div>Route root</div>} />
      <Route path="/app" render={()=><div>Route /app</div>} />
    </Switch> )
  export default App;
```

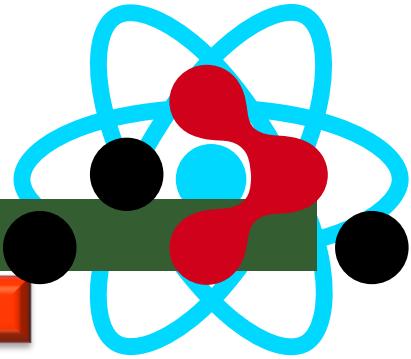


React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

Le fichier App.js

L'ajout du paramètre exact

Et d'une route par default une fausse 404

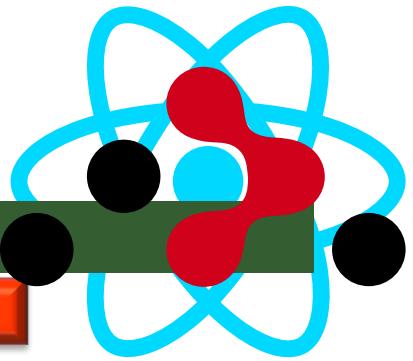
```
import React from "react";
import { BrowserRouter, Route, Link, Switch } from "react-router-dom";
import './App.css';
function App(props){
  return(
    <Switch>
      <Route exact path="/" render={()=><div>Route root</div>} />
      <Route path="/app" render={()=><div>Route /app</div>} />
      <Route render={()=><div>Route inconnue</div>} />
    </Switch> )
  export default App;
```

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

React Router

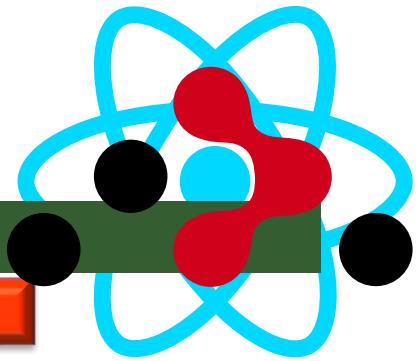
Le fichier App.js

```
<BrowserRouter>
  <div>
    <ul>
      <li><Link to="/">Home</Link></li>
      <li><Link to="/apropos">A propos</Link></li>
      <li><Link to="/sujets">Sujets</Link></li>
    </ul>
    <hr />
    <div className="main-route-place">
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/apropos" component={Apropos} />
        <Route path="/sujets" component={Sujets}>
          <Route component={Erreur} />
        </Switch>
      </div>
    </div>
  </BrowserRouter>
```

...

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

React Router

Le fichier App.js

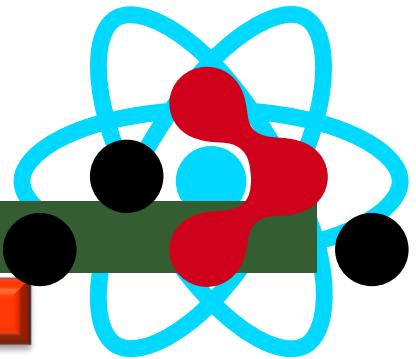
```
...
class Home extends React.Component {
  render() {
    return (
      <div>
        <h2>Home</h2>
      </div>
    );
  }
}
class Apropos extends React.Component {
  render() {
    return (
      <div>
        <h2>A Propos</h2>
      </div>
    );
  }
}
```

React.JS

un framework JavaScript

Application monopage

Javascript côté serveur



Utilisation du module react-router

Autre projet

Nous aurons besoin d'un module supplémentaire axios

npm install -g axios

Axios est un client HTTP basé sur des promesses qui arbore une API peut être utilisé à la fois dans le navigateur et Node.js.

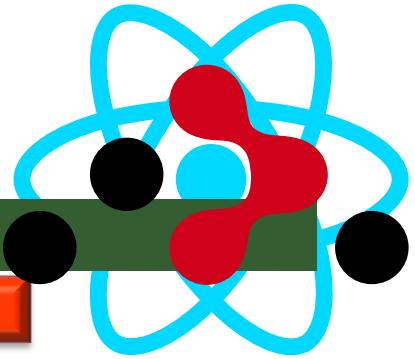
Faire des requêtes HTTP pour récupérer ou sauvegarder des données est l'une des tâches les plus courantes qu'une application JavaScript côté client devra effectuer. Les bibliothèques tierces – en particulier jQuery – ont longtemps été un moyen populaire d'interagir avec les API de navigateur plus détaillées et d'éliminer les différences entre les navigateurs

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

Autre projet

Nous aurons besoin d'un module supplémentaire axios

npm install -g axios

src/index.js

```
import React from 'react'
import { render } from 'react-dom'
import { BrowserRouter } from 'react-router-dom'
import App from './App'

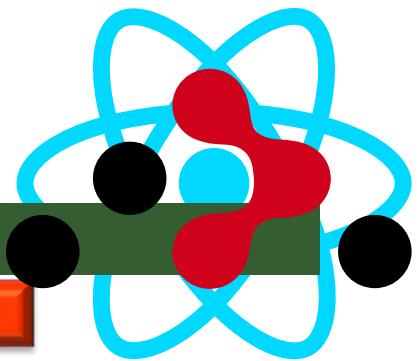
render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.querySelector('#root')
)
```

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module axios

Axios fait partie des clients HTTP les plus populaires basés sur les promesses pour les navigateurs et Node.js.

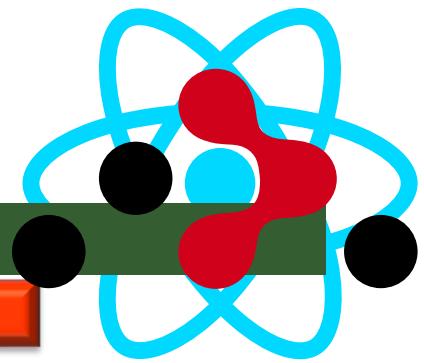
Axios fournit un support pour les intercepteurs de requête et de réponse, les transformateurs et la conversion automatique en JSON.

Cela vous protège également par défaut contre la falsification de requêtes intersites (XSRF).

`npm install -g axios`

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

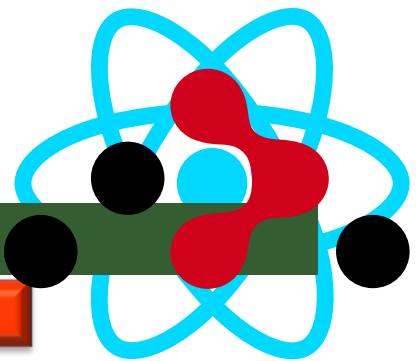
Utilisation du module axios

```
import React from 'react';
import axios from 'axios';

export default class PersonList extends React.Component
{
  state = {
    personnes: []
  }
  componentDidMount() { axios.get(`https://jsonplaceholder.typicode.com/users`)
    .then(res => {
      const personnes = res.data;
      this.setState({ personnes });
    })
  }
  render() {
    return (
      <ul>
        { this.state.personnes.map(personnes => <li>{personnes.email}</li>)}
      </ul>
    )
  }
}
```

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

Autre projet

Nous aurons besoin d'un module supplémentaire axios
src/App.js

```
import React from 'react'  
import { Route, Switch } from 'react-router-dom'  
import HomePage from './pages/HomePage'  
import UserPage from './pages/UserPage'  
export default function App() {  
  return (  
    <Switch>  
      <Route exact path="/" component={HomePage} />  
      <Route path="/:id" component={UserPage} />  
    </Switch>  
  )  
}
```

Dans le dossier src nous aurons besoin d'un répertoire "**pages**"

Et deux fichiers :

Pages/HomePage.js

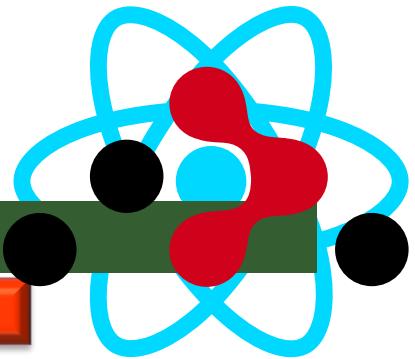
Pages/UserPage.js

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

Autre projet

Nous aurons besoin d'un module supplémentaire axios

src/pages/HomePage.js

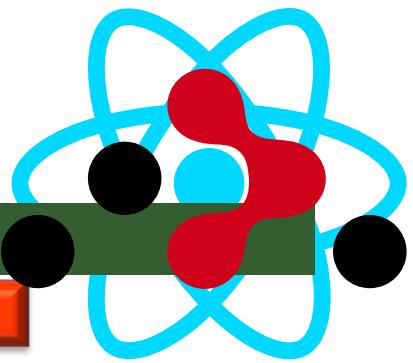
```
import React from 'react'  
import { Link } from 'react-router-dom'  
export default function HomePage() {  
  return (  
    <div className="container">  
      <h1>Home Page</h1>  
      <p>  
        <Link to="/react">Formation-react</Link> depuis Github  
      </p>  
    </div>  
  )  
}
```

React.JS

un framework JavaScript

Application monopage

Javascript coté serveur



Utilisation du module react-router

Autre projet

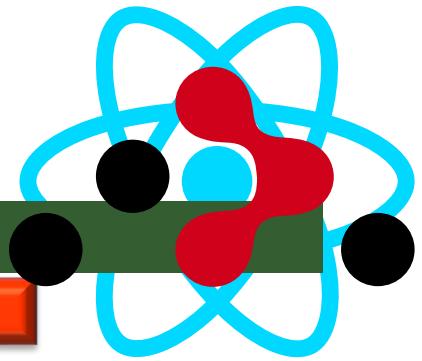
Nous aurons besoin d'un module supplémentaire axios

src/pages/UserPage.js

```
import React, { useState, useEffect } from 'react'
import axios from 'axios'
export default function UserPage(props) {
  const initialUserState = {
    user: {},
    loading: true,
  }
  const [user, setUser] = useState(initialUserState)
  useEffect(() => {
    const getUser = async () => {
      const { data } = await axios(`https://api.github.com/users/${props.match.params.id}`)
      setUser(data)
    }
    getUser()
  })
  ...
}
```

React.JS

un framework JavaScript



Application monopage

Javascript coté serveur

Utilisation du module react-router

Autre projet

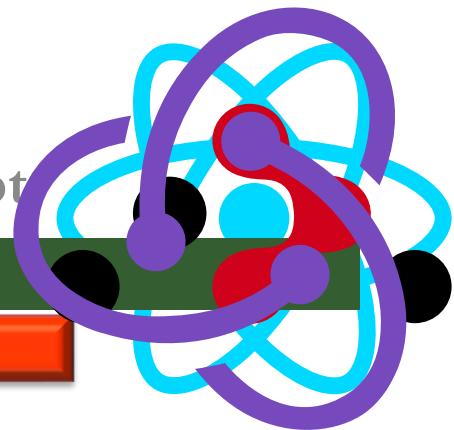
Nous aurons besoin d'un module supplémentaire axios

src/pages/UserPage.js

```
...
return user.loading ? (
  <div>Loading...</div>
) : (
<div className="container">
  <h1>{props.match.params.id}</h1>
  <table>
    <thead>
      <tr>
        <th>Id</th>
        <th>Name</th>
        <th>Location</th>
        <th>Website</th>
        <th>creation</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>{user.id}</td>
        <td>{user.name}</td>
        <td>{user.location}</td>
        <td> <a href={user.blog}>{user.blog}</a> </td>
        <td>{user.creat_at}</td>
      </tr>
    </tbody>
  </table>
</div>
)}
```

React.JS

un framework JavaScript

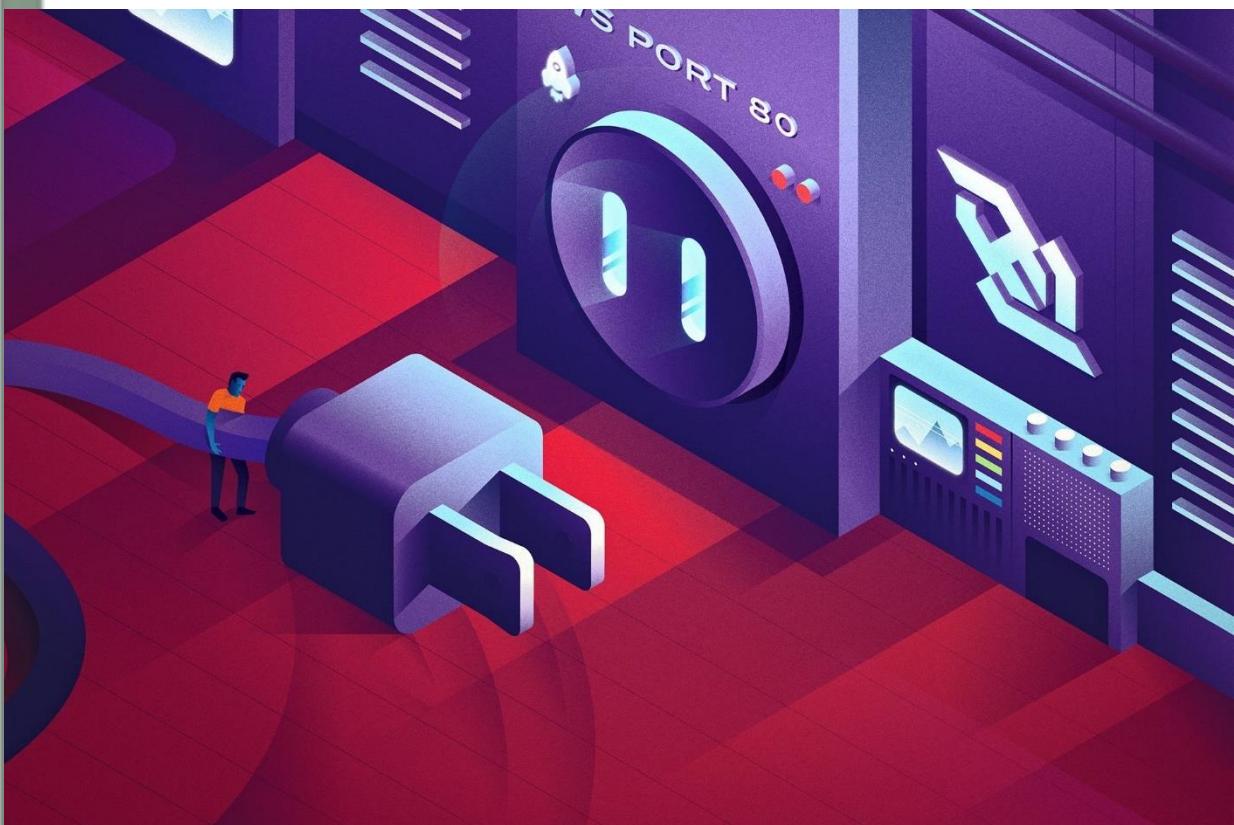


Application monopage

Projet Recapitulatif

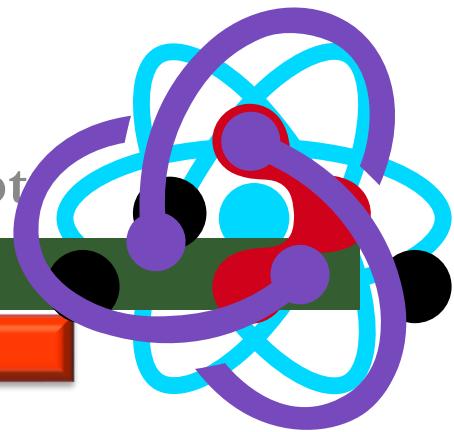
projet

Mise en place Serveur



React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

Optimiser l'application pour la production

Nous avons travaillé en mode développement mais on peut aussi lancer la commande

```
C:\nodejs\react-projet>npm run build
```

Le répertoire build est créé

```
OSX Invite de commandes
./src/App.js
Line 6: Useless constructor  no-useless-constructor
Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.

File sizes after gzip:
42.47 KB  build\static\js\2.1fbc0a.chunk.js
762 B     build\static\js\runtime~main.a8a9905a.js
701 B     build\static\js\main.ff33fd17.chunk.js

The project was built assuming it is hosted at the server root.
You can control this with the homepage field in your package.json.
For example, add this to build it for GitHub Pages:

  "homepage" : "http://myname.github.io/myapp",

The build folder is ready to be deployed.
You may serve it with a static server:

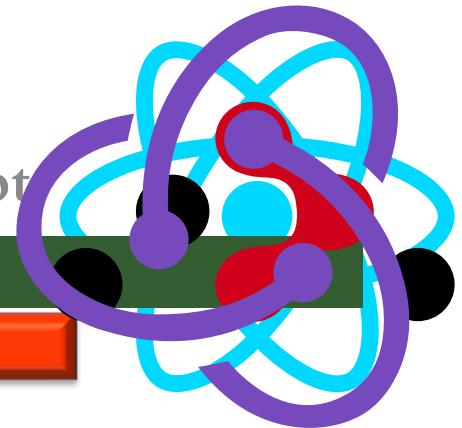
  npm install -g serve
  serve -s build

Find out more about deployment here:
https://bit.ly/CRA-deploy

C:\nodejs\reactapp>
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

Optimiser l'application pour la production

Installation et mise en place du serveur

C:\nodejs\react-projet>npm install -g serve

C:\nodejs\react-projet>npm run build

C:\nodejs\react-projet> serve build

```
cmd Invite de commandes - serve -s build
serve -s build
Find out more about deployment here:
https://bit.ly/CRA-deploy

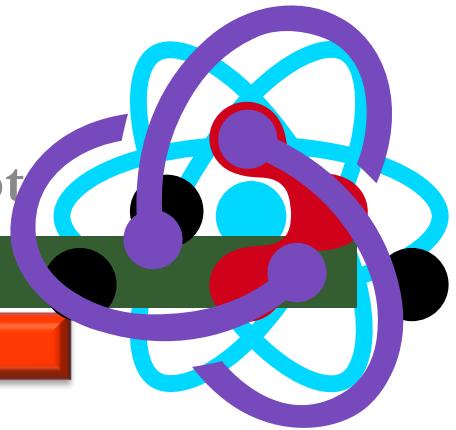
C:\nodejs\reactapp>npm install -g serve
C:\Users\Stéphane\AppData\Roaming\npm\serve -> C:\Users\Stéphane\AppData\Roaming\npm\node_modules\serve\bin\serve.js
+ serve@11.0.0
added 78 packages from 39 contributors in 25.962s

C:\nodejs\reactapp>server -s build
C:\nodejs\reactapp>server -s build
C:\nodejs\reactapp>serve -s build
:
:
:
Serving!
- Local:          http://localhost:5000
- On Your Network: http://192.168.1.113:5000
Copied local address to clipboard!
```

http://localhost:5000

React.JS

un framework JavaScript



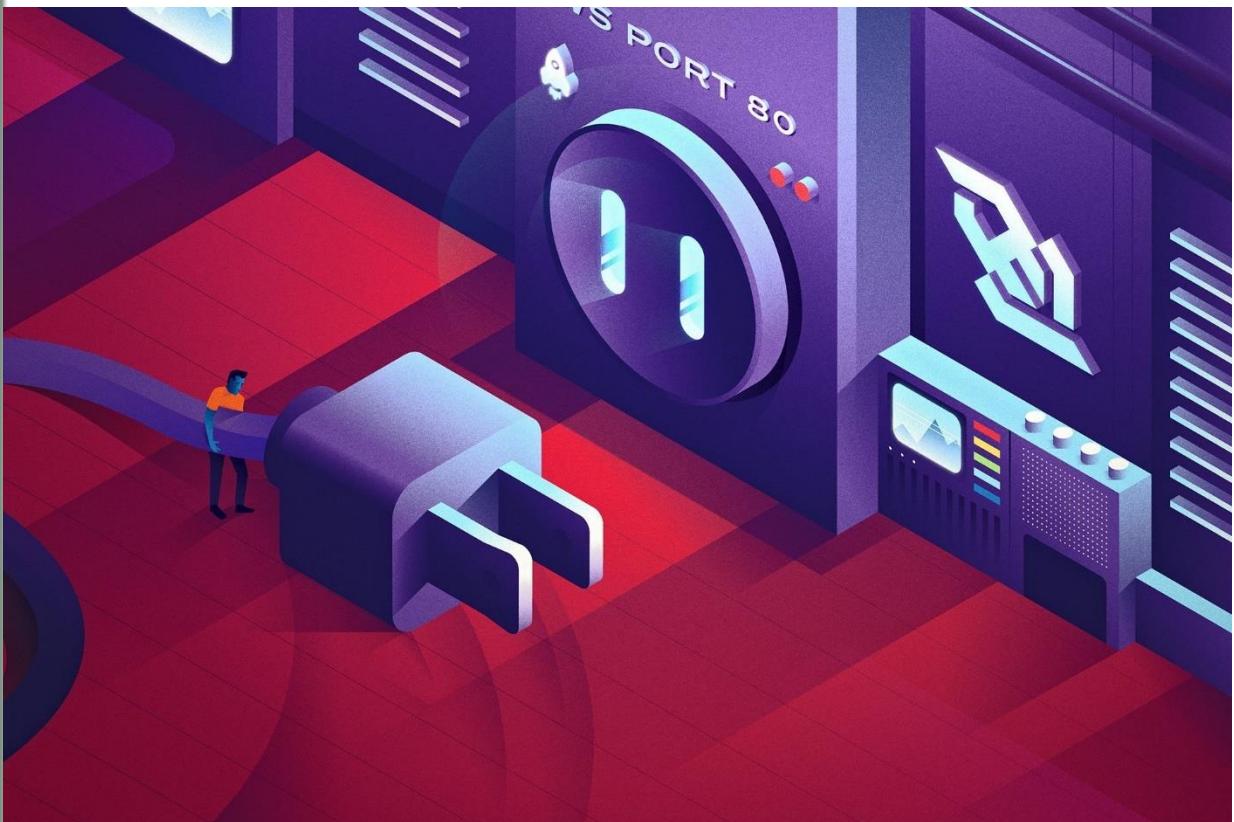
Application monopage

Projet Recapitulatif

Optimiser l'application pour la production

`http://localhost:5000`

Le serveur en mode production la page s'affiche plus rapidement

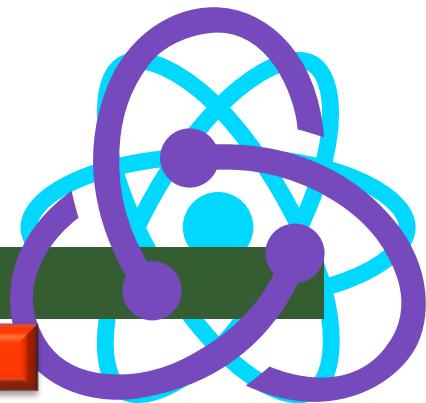


React.JS

un framework JavaScript

Application monopage

Redux



Redux

Dans **React**, on distingue deux types de données :

Les *propriétés* (props) qui sont accessibles uniquement en lecture seule

L'*état* (state) qui est disponible en lecture/écriture mais qui reste propre à un composant (local).

La modification du *state local* ou la réception de nouvelles propriétés (mise à jour du composant) entraîne un rafraîchissement de l'**UI**.

Avec **Redux**, on ajoute la *notion de store*.

Un *store* est global. Il contient toutes les informations de tous les composants qui y sont rattachés.

Lorsqu'un composant **A** lance une action et entraîne une modification du store (dit aussi “*state global*”), le composant **B** aura également les nouvelles données du composant **A**.

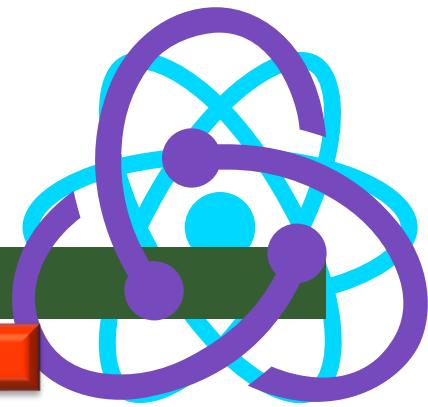
Chaque composant peut mettre en place des comportements différents en fonction de l'état des autres.

Le store va également contenir les actions disponibles dans les composants.

.

React.JS

un framework JavaScript



Application monopage

Redux

Redux

Redux est une bibliothèque JS permettant de gérer l'état d'une application de manière déterministe.

Redux propose un conteneur (le store) dont les modifications sont décrites par des actions (sortes d'évènements) qui sont gérés par un réducteur (reducer).

Dans les cas suivants, nous pouvons utiliser redux.

Même élément de données à plusieurs endroits

Plusieurs vues qui doivent fonctionner avec les mêmes données en synchronisation

Les données peuvent être modifiées par plusieurs actions / plusieurs utilisateurs

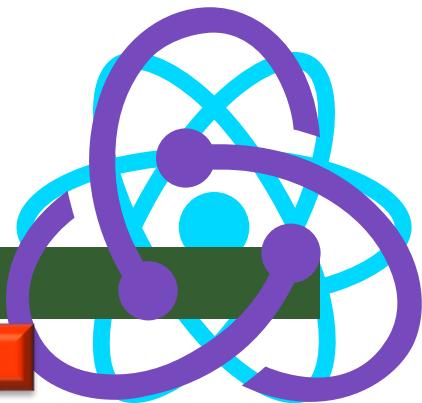


React.JS

un framework JavaScript

Application monopage

Redux



Redux les actions

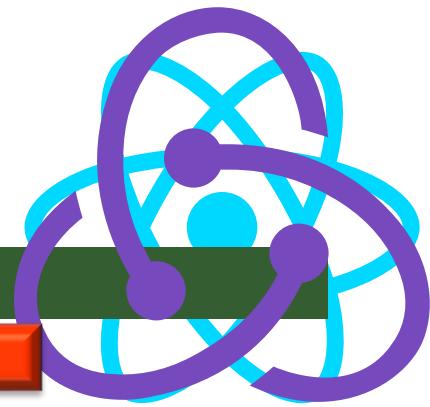
Une action est un simple objet JS qui a pour seule contrainte d'avoir une propriété type sérialisable (un string) ainsi que n'importe quelles autres propriétés permettant au réducteur de générer un nouvel état.

l'action contient un verbe actif. Il doit permettre d'identifier la nature de la modification correspondant à l'action

```
// déclaration de l'actions
export function decr_time({min, sec}){
  return {
    type : ACTIONS.DECR_TIME,
    time : {min, sec}
  }
}
```

React.JS

un framework JavaScript



Application monopage

Redux

Redux les reducers

Un réducteur (reducer) est une fonction pure, qui prend en paramètre un état et une action, pour retourner un (nouvel) état
Décrit comment une action va modifier un état donner pour retourner un nouvel état.

Le nouvel état, retourné par la fonction, est un nouvel objet.
L'état d'origine n'est pas modifié.

Cette fonction est dite pure car :

elle est déterministe : si on l'appelle 10 fois avec les mêmes paramètres, elle renvoie 10 fois le même résultat ;

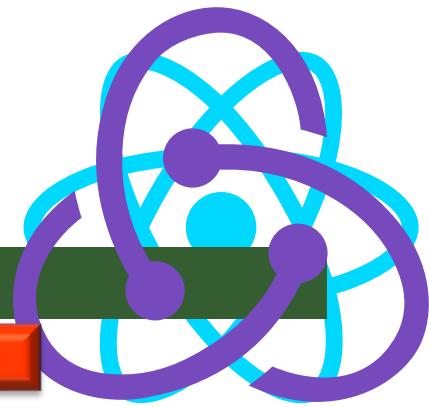
elle ne modifie pas ses paramètres ;

Elle n'a pas d'effet de bord elle ne modifie aucune ressource externe (pas de fermeture).

```
export default function reducer(state = stateInit, action){  
  var newState;  
  if (action.type==ACTIONS.DECR_TIME) {  
    var time = decrTime(state.time);  
    newState={ ...state, time};  
  }  
  else newState=state  
  return newState;  
}
```

React.JS

un framework JavaScript



Application monopage

Redux

Redux le store

Le store redux est l'objet javascript qui contient l'état immuable d'une application.

Toute modification du store doit passer par un reducer qui va générer un nouvel état.

On crée un store avec la fonction createStore et en paramètre un réducteur capable de gérer les actions.

```
const store = createStore(reducer);

setInterval(function(){
    var time = store.getState().time; //enlever 1sec
    store.dispatch(ACTIONS.decr_time(time));
}, 1000);
```

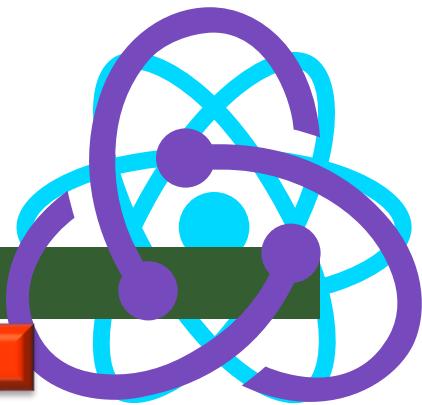


React.JS

un framework JavaScript

Application monopage

Redux



Redux le store

L'objet store possède trois méthodes :

subscribe qui permet à tout écouteur (listener) d'être notifié en cas de modification du store. Les gestionnaires de vues (comme React) vont souscrire au store pour être notifié des modifications et effectuer mettre à jour l'interface graphique en conséquence.

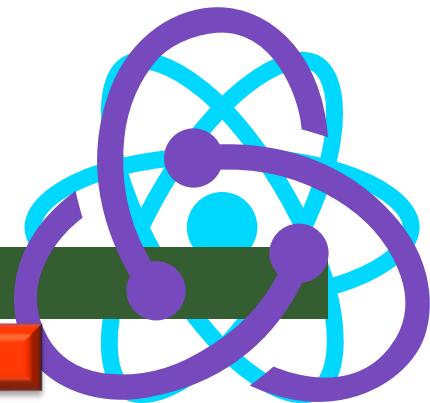
dispatch qui prend en paramètre une action et exécute le reducer qui va, à son tour, mettre à jour le store avec un nouvel état.

getState qui retourne l'état courant du store. L'objet retourné ne doit pas être modifié.



React.JS

un framework JavaScript



Application monopage

Redux

Redux le store

Pour donner accès au store au différents composants d'une application React, Redux offre le composant **Provider**.

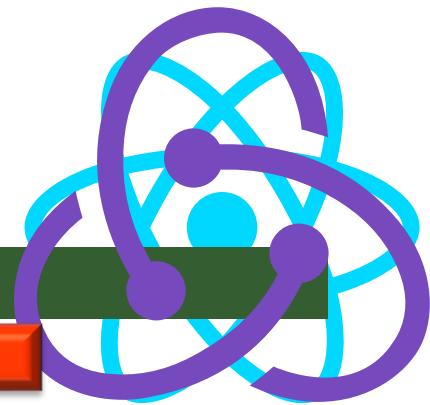
```
import reducer from './reducer.js';
import * as ACTIONS from './actions.js';

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>
  ,document.querySelector("#root")
);
```



React.JS

un framework JavaScript



Application monopage

Redux

Redux le store

Pour qu'un composant React ait accès au store, il faut "connecter" ce dernier au store.

On utilise la fonction **connect** de Redux sur le composant désiré pour lui donner accès.

La fonction **connect** retourne un nouveau composant React ayant la possibilité de

- faire un **getState()** sur le store pour lire l'état courant;
- d'appeler **dispatch()** pour déclencher la génération d'un nouvel état.

Les options de la fonction **connect** sont nombreuses.

On peut vouloir avoir accès à la fonction dispatch uniquement ou bien s'enregistrer pour recevoir les modifications du store

```
import { connect } from 'react-redux';
```

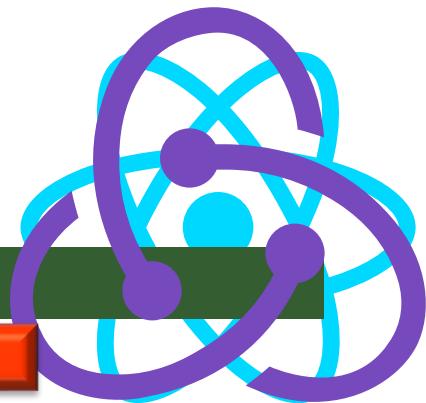
```
export default connect(mapStateToProps)(App);
```

React.JS

un framework JavaScript

Application monopage

Redux



Redux le store

on définit deux fonctions :

mapStateToProps pour définir des propriétés (props) dans le composant à partir des valeurs du store.

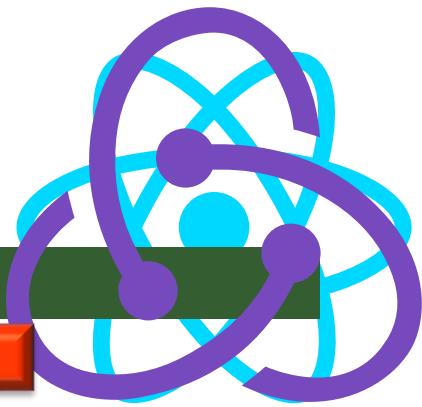
mapDispatchToProps pour définir des fonctions de callback dans les props qui vont être bindées avec la fonction dispatch et vont faire appel au reducer.

React.JS

un framework JavaScript

Application monopage

Redux



Redux

Le reducers c'est un peu la fonction callback de redux

Le reducers vous permet de gérer la mise à jour des données.

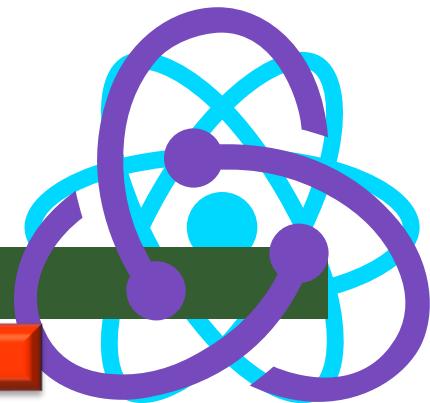
Le reducers et les actions sont complémentaires.

Les traitements comme les requêtes sont réalisées dans le fichier actions.js et le reducers s'occupent de mettre à jour les données du store (state global).



React.JS

un framework JavaScript



Application monopage

Redux

Installation de Redux

Nous allons l'installer dans le projet reactapp

C:\nodejs> reactapp> npm install redux

Nous allons modifier le fichier src/index.js

En important la fonction createStore()

```
import {createStore} from "redux";
var store = createStore(function(state, action){
    return state;
});
console.info(store);
```

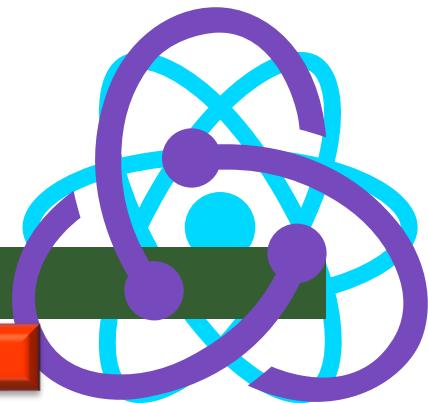
```
Download the React DevTools for a better development experience: https://fb.me/react-devtools
{...}
  ▶ dispatch: function dispatch() {
  ▶ getState: function getState() {
  ▶ replaceReducer: function replaceReducer() {
  ▶ subscribe: function subscribe() {
  ▶ Symbol(observable): function observable() {
  ▶ <prototype>: Object { ... }
```

React.JS

un framework JavaScript

Application monopage

Redux



Utilisation de react et redux

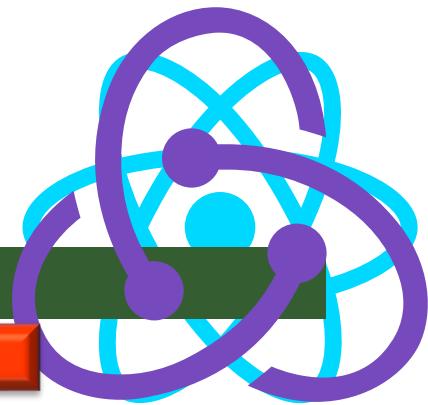
On a utilisé React pour la partie vue des applications web .

Redux permet de gérer la partie données et contrôler.

Il est aussi possible d'utiliser **Redux** avec d'autres Frameworks comme **Angular** ou **Ember**

React.JS

un framework JavaScript



Application monopage

Redux

Utilisation de react et redux

En import

```
//liste des actions types
const HANDLE_CHANGE = 'HANDLECHANGE';

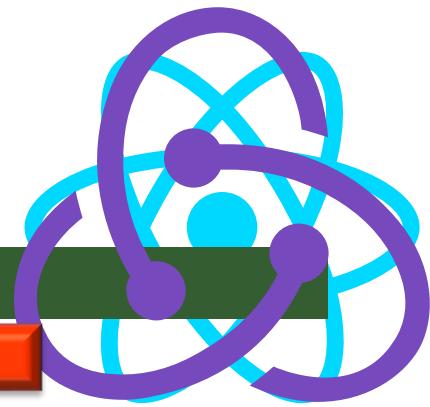
//initialisation des valeurs
const defaultState = {
  firstName: "John",
  lastName: "Doe",
  chairs: true,
  tables: true };
//initialisation des valeurs

//actions
const handleChange = event => {
  const { name, value, type } = event.target;
  return {
    type: HANDLE_CHANGE,
    value,
    name,
    input: type };

};
```

React.JS

un framework JavaScript



Application monopage

Redux

Utilisation de react et redux

En import

```
const formReducer = (state = defaultState, action) => {
  switch (action.type) {
    case HANDLE_CHANGE:
      let newState = Object.assign({}, state);
      action.input === "checkbox" ? newState[action.name] = !newState[action.name] : newState[action.name] = action.value;
      return newState;
    default:
      return state;
  };
  //mise a jour ou non du store
  const store = Redux.createStore(formReducer);

  //React-Redux

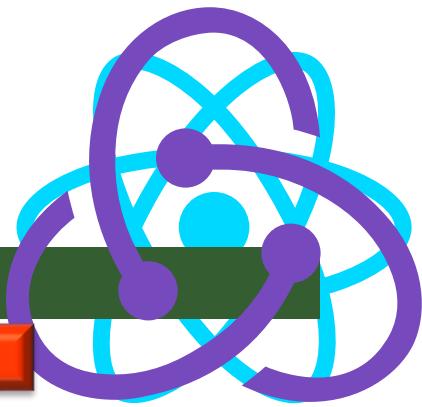
  //mapStateToProps, fonctions qui parcourir le state de l'application et les actions
  //et qui vont mettre à jour le store

  const mapStateToProps = state => {
    return {
      firstName: state.firstName,
      lastName: state.lastName,
      chairs: state.chairs,
      tables: state.tables };
  };
}
```



React.JS

un framework JavaScript



Application monopage

Redux

Utilisation de react et redux

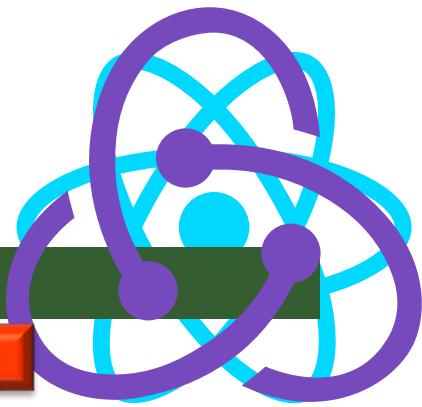
En import

```
const mapDispatchToProps = dispatch => {
  return {
    handleChange: event => {
      dispatch(handleChange(event));
    }
  };
};

//React
const Form = props => {
  return (
    React.createElement("div", null,
      React.createElement("form", null,
        React.createElement("input", { type: "text", name: "firstName", value: props.firstName, onChange: () => props.handleChange("firstName")}),
        React.createElement("input", { type: "text", name: "lastName", value: props.lastName, onChange: () => props.handleChange("lastName")}),
        React.createElement("input", { type: "checkbox", name: "chairs", checked: props.chairs, onChange: () => props.handleChange("chairs")}),
        React.createElement("input", { type: "checkbox", name: "tables", checked: props.tables, onChange: () => props.handleChange("tables")}),
        React.createElement("button", { type: "submit" }, "Submit"),
        React.createElement("h4", null, props.tables ? 'Tables' : " "),
        React.createElement("h4", null, props.chairs ? 'Chairs' : " "),
        React.createElement("p", null, props.lastName ? props.lastName : " "),
        React.createElement("p", null, props.firstName ? props.firstName : " ")
      )));
};
```

React.JS

un framework JavaScript



Application monopage

Redux

Utilisation de react et redux

En import

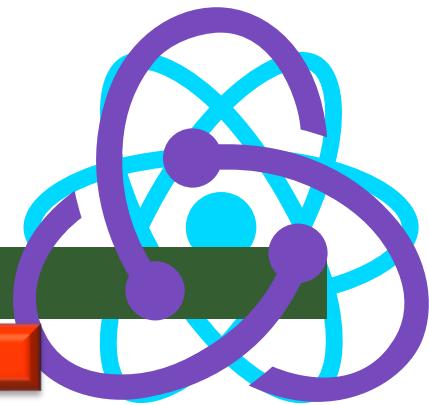
```
const connect = ReactRedux.connect;
const Provider = ReactRedux.Provider;
const ConnectedComponent = connect(mapStateToProps, mapDispatchToProps)(Form);
const AppWrapper = () => {
  return (
    React.createElement(Provider, { store: store },
      React.createElement(ConnectedComponent, null)));
};

ReactDOM.render(React.createElement(AppWrapper, null), document.querySelector('#root'));
```



React.JS

un framework JavaScript



Application monopage

Redux

Utilisation du module react-redux

Il existe un package **react-redux** pour que les deux solutions React et Redux puissent communiquer entre elles

C:\nodejs> reactapp> npm install react-redux

```
C:\nodejs\reactapp>npm install react-redux
npm WARN ts-pnp@1.0.1 requires a peer of typescript@* but none is installed. You must install peer dependencies
.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","ar
"} (current: {"os":"win32","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","ar
"} (current: {"os":"win32","arch":"x64"})

+ react-redux@7.0.2
added 4 packages from 4 contributors and audited 36272 packages in 89.277s
found 68 vulnerabilities (63 low, 5 high)
  run `npm audit fix` to fix them, or `npm audit` for details

C:\nodejs\reactapp>
```

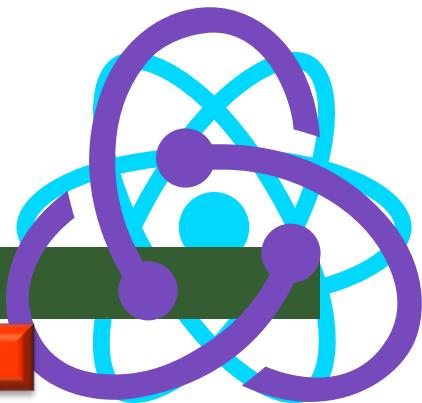


React.JS

un framework JavaScript

Application monopage

Redux



Utilisation du module react-redux

Le module "react-redux" comporte la méthode connect() qui facilite l'accès au store, pour lire et effectuer des actions pour le mettre à jour

Nous devons toujours créer le store via createStore()

Il faut aussi créer un composant parent <Provider>

Le fichier index.js devient

```
import React from "react";
import { render } from "react-dom";
import { Provider } from "react-redux";
import store from "./js/store/index";
import App from "./js/components/App";

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.querySelector("#root")
);
```

React.JS

un framework JavaScript

Application monopage

Redux

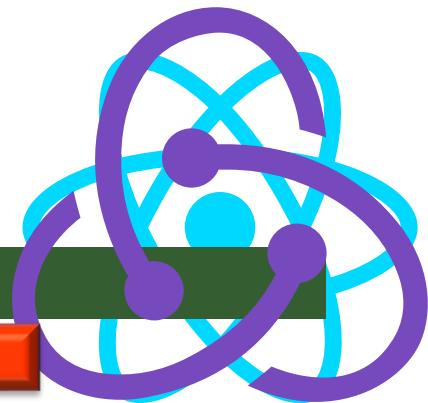
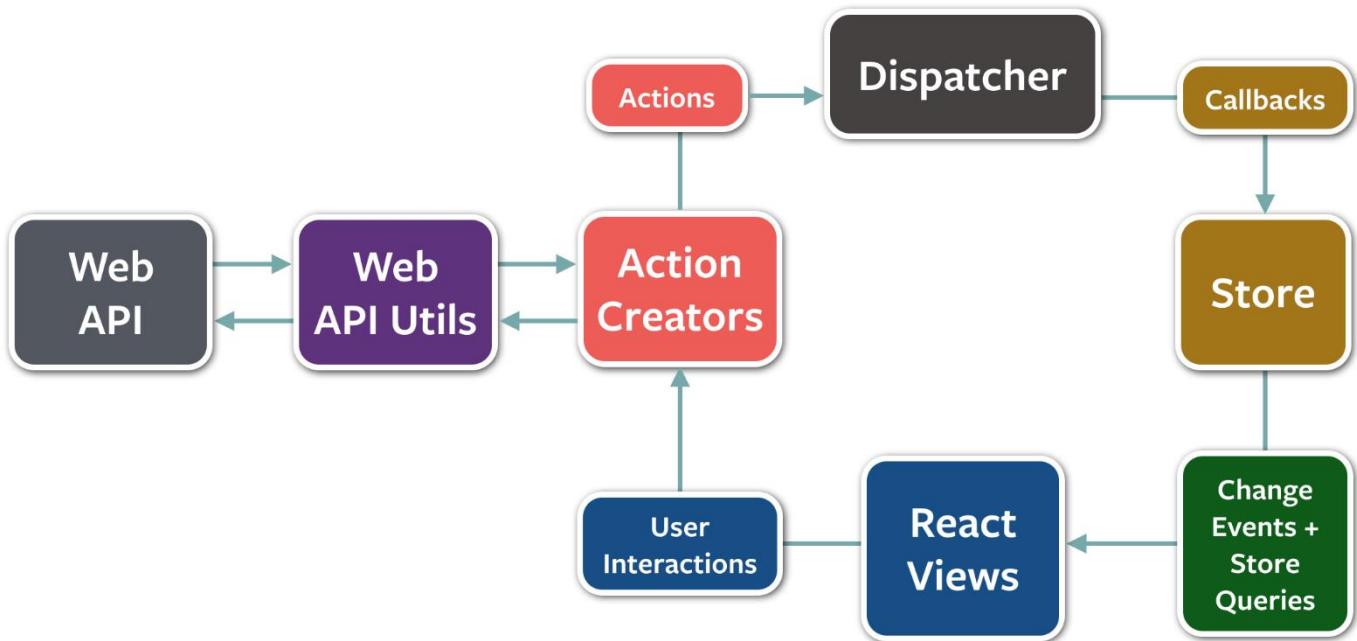
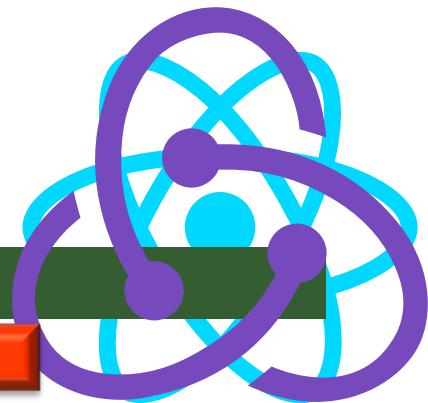


Schéma des étapes des évènements dans react



React.JS

un framework JavaScript



Application monopage

Redux

Actions

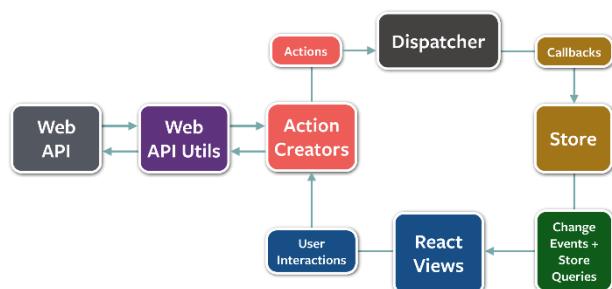
En rouge sur le schéma.

Dans une architecture Flux, tout passe par les actions.

On ne peut pas modifier l'affichage d'un composant ou déclencher un comportement sans action.

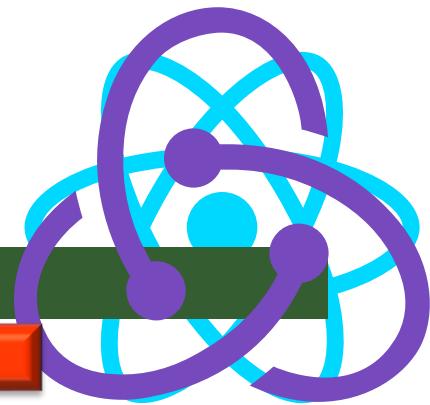
C'est à partir d'une action qu'on pourra modifier le state d'un composant React par exemple.

Si le code de l'action est complexe,
on peut séparer sa création et l'instance elle-même



React.JS

un framework JavaScript



Application monopage

Redux

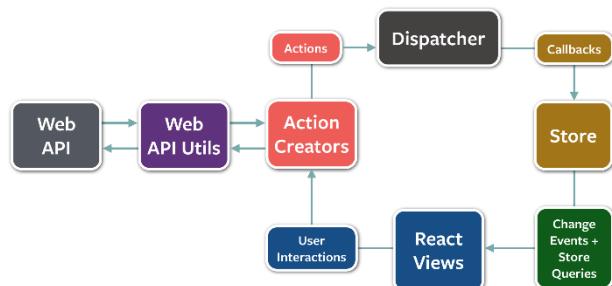
Reducer/dispatcher

En noir sur le schéma.

Les reducers : sont des fonctions pures qui prennent en arguments un state et une action, et qui retournent le state après que l' action ait été effectuée.

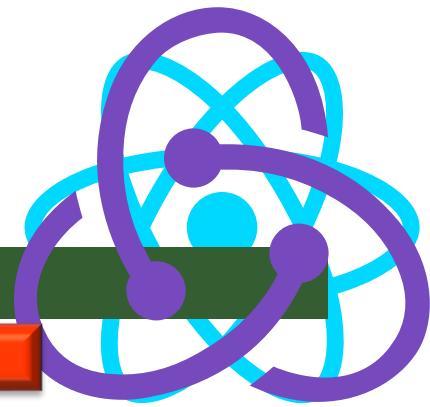
Le dispatcher est le composant unique qui reçoit toutes les actions de l'application.

Son rôle est de notifier tous les stores via des callbacks que l'action a eu lieu



React.JS

un framework JavaScript



Application monopage

Redux

Les stores

En marron sur le schéma.

Les stores sont les composants de Flux qui vont contenir et gérer les states de l'application.

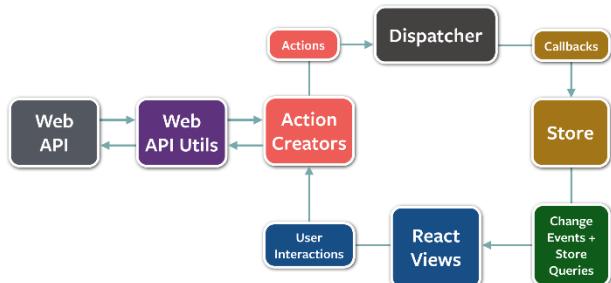
Ils fournissent au dispatcher les callbacks exécutés lors de la notification d'une action.

Il va donc contenir l'implémentation de toutes les règles de gestions du domaine qu'il couvre.

Il va également gérer les actions qu'il veut traiter, car comme expliqué ci-dessus, le dispatcher notifie les stores de toutes les actions de l'application.

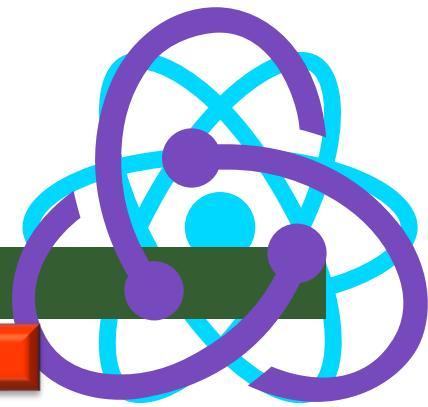
Chaque store s'occupera d'une partie du fonctionnel de l'application et ne voudra donc pas traiter toutes les actions.

Pour finir, les stores vont notifier par événement les changements d'état aux vues leur correspondant.



React.JS

un framework JavaScript



Application monopage

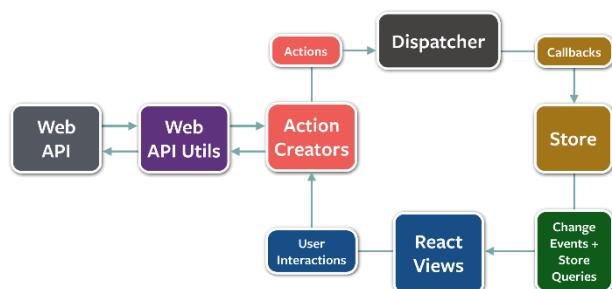
Redux

Les vues

En bleu sur le schéma.

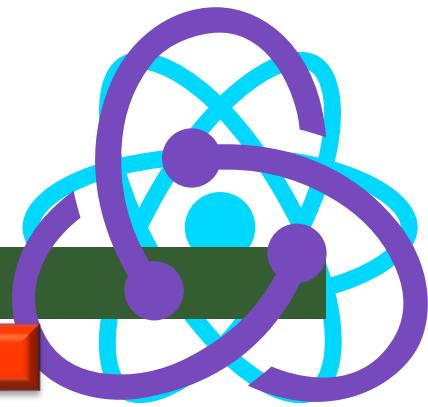
Les vues sont chargées d'afficher le state contenu dans le store associé.

Elles s'enregistrent auprès du store pour être notifiées des changements de state, pour se mettre à jour en conséquence.



React.JS

un framework JavaScript



Application monopage

Redux

store est le résultat de createStore,
qui est à son tour une fonction de la bibliothèque redux.
createStore prend un reducer comme premier argument .

Vous pouvez passer un état initial à createStore, ce qui est utile pour le rendu côté serveur.

Le concept le plus important ici est que **l'état dans redux provient des Reducers.**

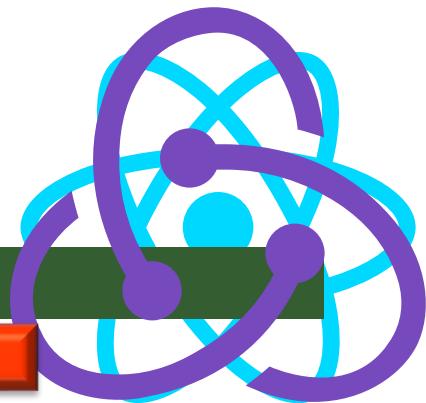
Les Reducers produisent l'état de votre application

React.JS

un framework JavaScript

Application monopage

Redux



Un reducer est juste une fonction JavaScript .

Un reducer prend deux paramètres : l'état actuel et une **action**.

Dans un composant React typique, l'état local change avec `setState`.

Dans Redux, vous ne pouvez pas faire cela.

Le troisième principe de Redux dit que l'état est immuable et qu'il ne peut pas changer de place.

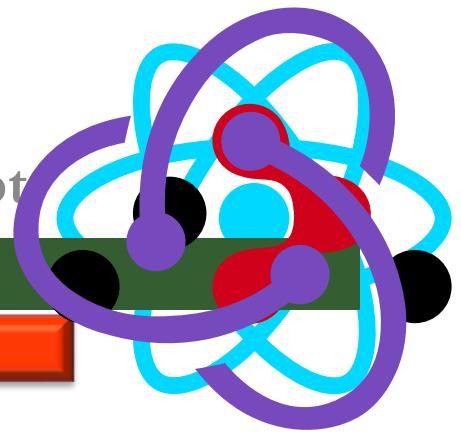
seul moyen de changer l'état consiste à envoyer un signal au store .

Ce signal est une **action**



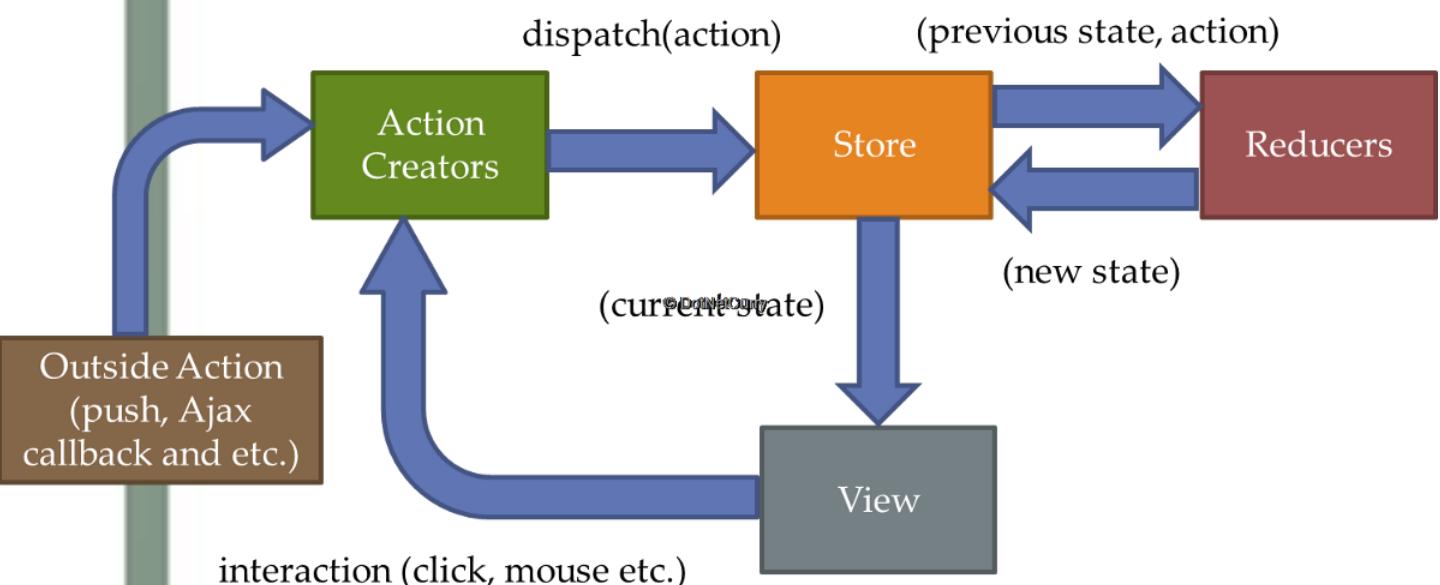
React.JS

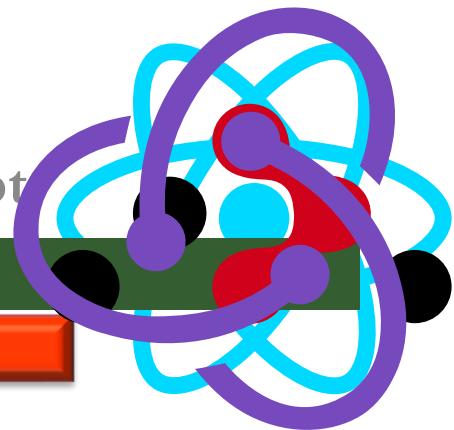
un framework JavaScript



Application monopage

Projet Recapitulatif





Application monopage

Projet Recapitulatif

Redux

Store : objet javascript qui va être persistent dans le temps.

On va stocker les données de l'application qui vont devoir persister pendant toute l'utilisation.

Il peut être stocké dans le local storage pour durer plus longtemps

Reducer : c'est une fonction qui a le droit d'écrire dans le store.

On appelle cette fonction, avec en argument un nom d'événement, exemple : *un setter de username avec le username*.

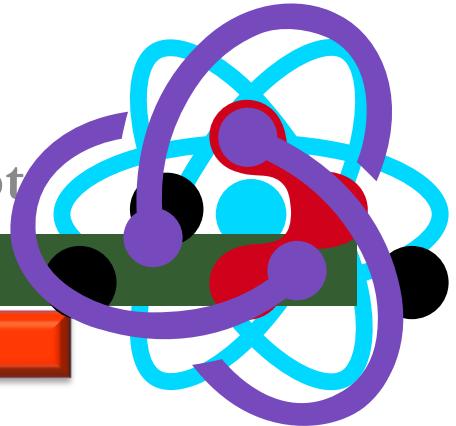
Cette fonction va mettre à jour la donnée dans le store.

Dans la **vue**, on déclenche un évènement, ensuite un reducer va réagir à cet évènement, comme un listener, et ensuite le store va être mis à jour.

React-Redux: quand le store est à jour, on va pouvoir mettre à jour le composant.

React.JS

un framework JavaScript



Application monopage

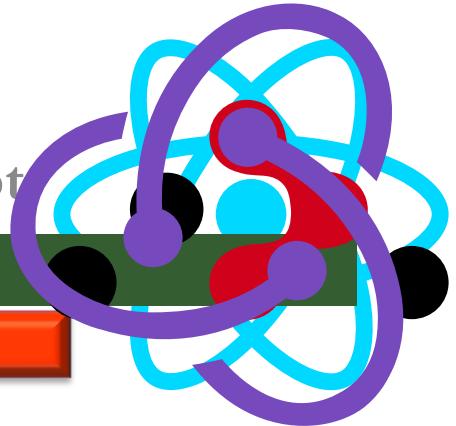
Projet Recapitulatif

projet

Avec ce projet nous allons utiliser React-Redux et montrer comment échafauder un projet basé sur react-redux à partir de create-react-app

```
C:\nodejs>npm install -g create-react-app  
C:\nodejs>create-react-app react-projet  
C:\nodejs>cd react-projet  
C:\nodejs\react-projet>npm install react-redux  
C:\nodejs\react-projet>npm install react-router-dom
```





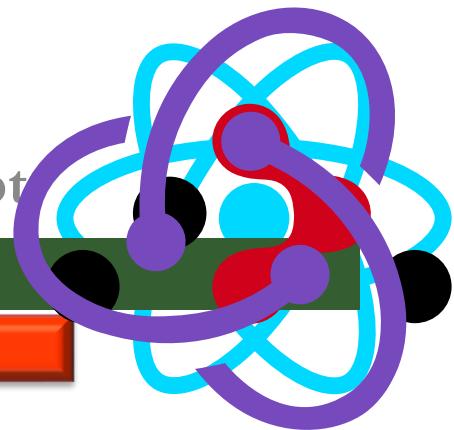
Application monopage

Projet Recapitulatif

projet

Avec ce projet nous allons utiliser React-Redux et montrer comment échafauder un projet basé sur react-redux à partir de create-react-app

```
"dependencies": {  
  "react": "^16.8.6",  
  "react-dom": "^16.8.6",  
  "react-redux": "^7.0.2",  
  "react-router-dom": "^5.0.0",  
  "react-scripts": "2.1.8"  
},
```



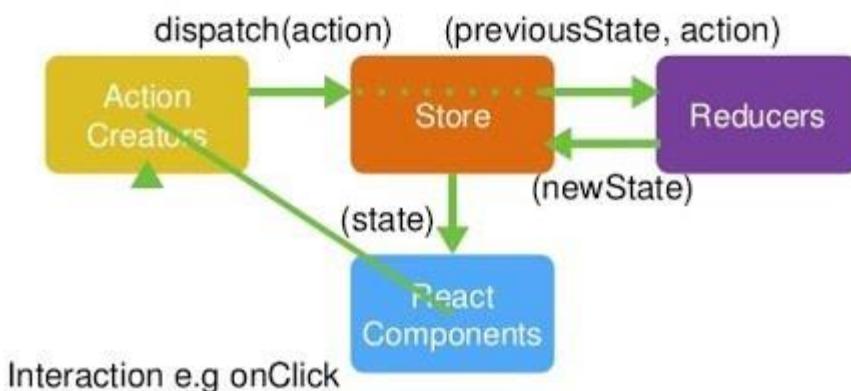
Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Redux Flow

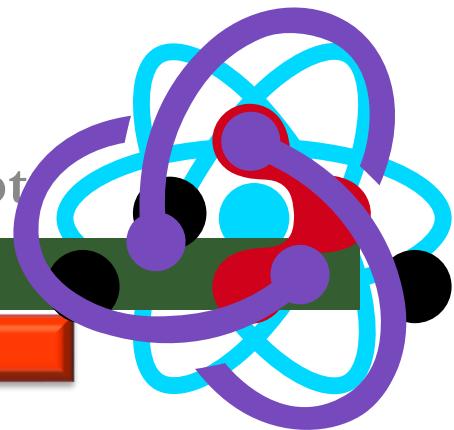


le flux de travail comporte quatre parties: **store** , **actions** , **composants** , **reducers** , tandis que React-Redux sépare les composants en deux: **composants** et **conteners** pour la conception d'interface utilisateur et la logique fonctionnelle, respectivement.



React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

le store

Un fichier pour stockez les états, c'est un peu comme un entrepôt.

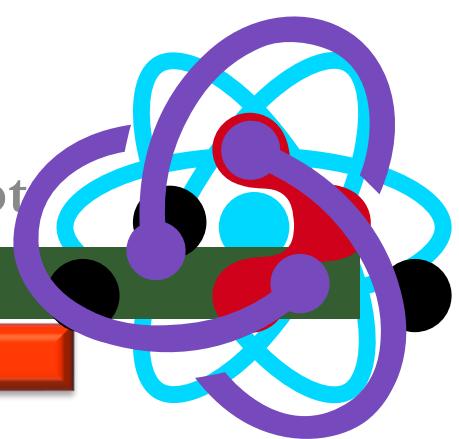
Les applications n'ont qu'un seul Store.

Components

Le modèle d'interface utilisateur.

Les containers

Les états gérés pour chaque composant fournissent également des méthodes pour générer des actions.



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Reducers

Après avoir reçu une action, votre magasin doit fournir un nouvel ensemble d'états pour que la nouvelle vue soit côté client.

Les réducteurs jouent le rôle de reconnaître le type d'actions et de traiter les données, ainsi que de générer les nouveaux états.

Nous allons maintenant créer des répertoires tels que

src / actions ,

src / components ,

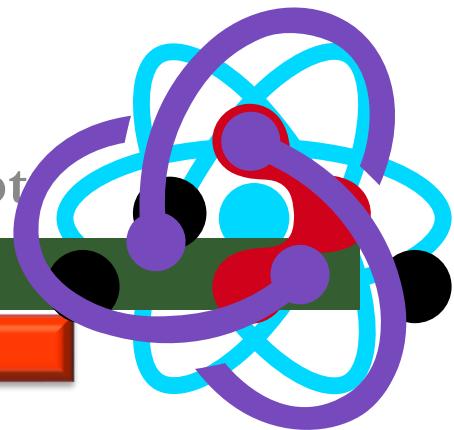
src / containers ,

src / reducers , et n'oubliez pas

src / store.js

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

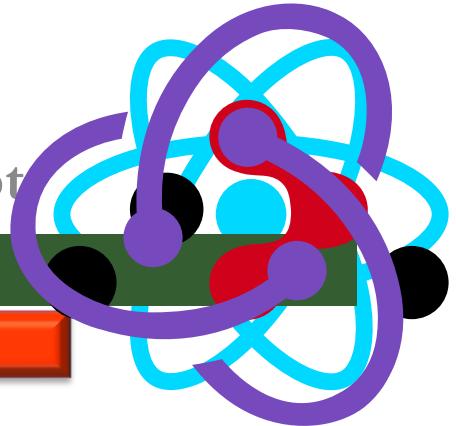
L'organigramme Redux

le fichier **index.js** est le point de départ qui permet d'accéder à tous les autres fichiers.

C'est un fichier minimaliste qui dépasse rarement les 30 lignes ; le dossier **components** contient tout ce que vous avez déjà fait dans la première partie, c'est-à-dire des composants "simples", dits "presentational" ou "stateless" ;

les dossiers **actions** et **constants** vont de pair et définissent les actions, comme présenté dans le premier chapitre ; le dossier **reducers** contient tous les reducers, y compris celui qui regroupera les différents reducers (via la méthode `combineReducers()`) ;

le dossier **containers App.js** contient les containers, aussi appelés "smart components" ou "stateful components". Ce sont des components qui communiqueront directement avec la partie Redux du code, en récupérant les states et qui vont dispatch les actions



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Combiner les réducteurs

La fonction `createStore` ne prend en paramètre qu'un seul réducteur qui va être chargé de gérer toutes les actions de l'application.

Or plusieurs types d'actions différentes vont cohabiter.

Par exemple, les actions liées à la modification du model peuvent être séparées des actions liées à l'interface graphique.

On va combiner les réducteurs avec la fonction **combineReducers**.

Exemple

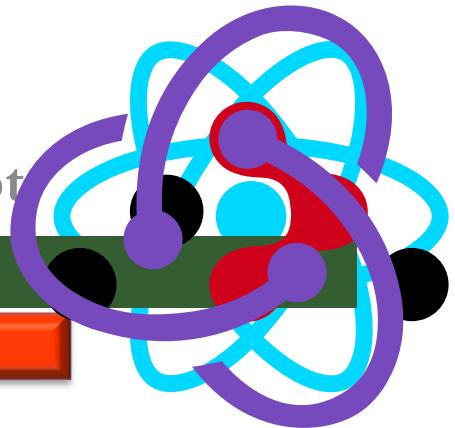
On stocke 2 types d'informations dans le store :

Un tableau d'objets (`stuff`) qui représente le modèle,

Une propriété graphique (`display`) qui permet de savoir comment afficher les objets (en liste, en grille, en miniatures...)

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Combiner les réducteurs

```
const stuff = (state, action) => {
  switch(action.type){
    'ADD_STUFF': return [
      ...state,
      {id: action.id, stuff: action.stuff}
    ]
    'REMOVE_STUFF': return state.filter((s)=>(s.id !== action.id))
    default: return state
  }
}

const display = (state, action) => {
  switch(action.type){
    'CHANGE_DISPLAY': return action.displayType
    default: 'DISPLAY_LIST'
  }
}

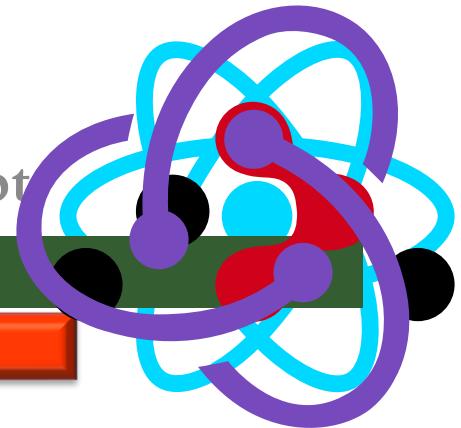
const reducer = combineReducers({
  stuff,
  display
})

const initialState = {
  stuff: [
    { id:1, stuff:'OK'},
    { id:2, stuff:'KO'}
  ],
  display: 'DISPLAY_LIST'
}

const store = createStore(reducer, initialState)
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Reducers

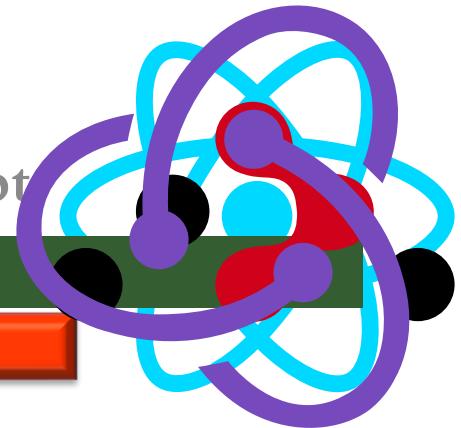
un CountReducer qui fournit des méthodes de traitement des actions pour les répartiteurs

src / reducers /CountReducer.js

```
export default function CountReducer(state = {  
  count: 0, wish_value: 0,}, action) {  
  const count = state.count const wish_value = action.wish_value switch  
(action.type) {  
    case 'increase':  
      return {count: Number(count) + 1}  
    case 'update':  
      return {  
        count: wish_value, }  
    default:  
      return state  
  }  
}
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

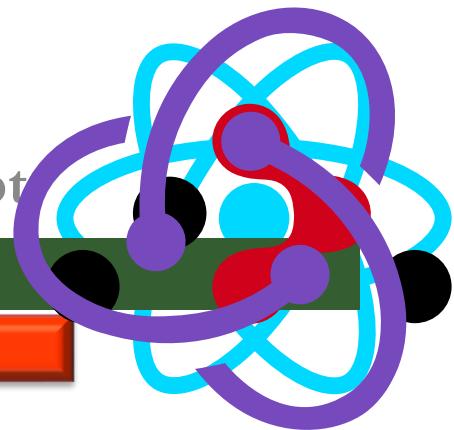
Reducers

créer un fichier index dans src / reducers / index.js afin de combiner tous les réducteurs contenus dans le dossier src / reducers.

Vous pourrez charger tous les reducers à la fois et ne jamais changer de store.js

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

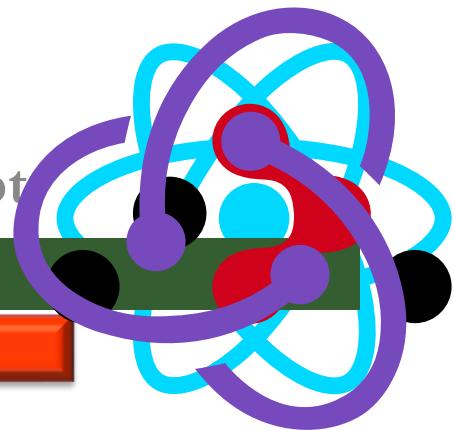
Actions/index.js

Création des constantes actions

```
//Action Creator
export const increaseTodo = () => {
  return {
    type: 'increase',
  }
}
export const updateTodo = (wish_value) => {
  return {
    type: 'update',
    wish_value: wish_value,
  }
}
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

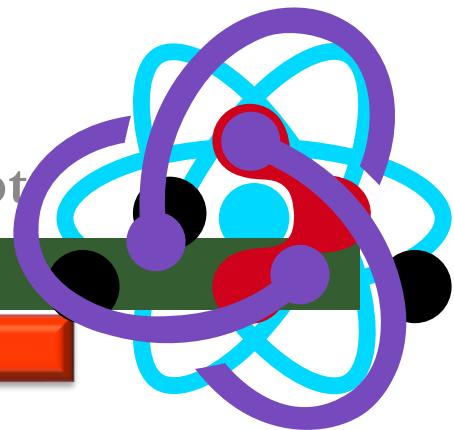
L'organigramme Redux

Créez un store exportez, qui seront importés dans
<fournisseur>, store.js

```
import {createStore} from 'redux'  
import {combineReducers} from 'redux'  
import CountReducer from './reducers/CountReducer'  
const reducer = combineReducers({  
    CountReducer  
});  
const initialState = {  
    CountReducer: {count: 123, wish_value: 12}  
};  
let store = createStore(reducer, initialState);  
export default store;
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Définissez le provider sur **src/ index.js** avec un objet store importé sous la forme **store.js**.

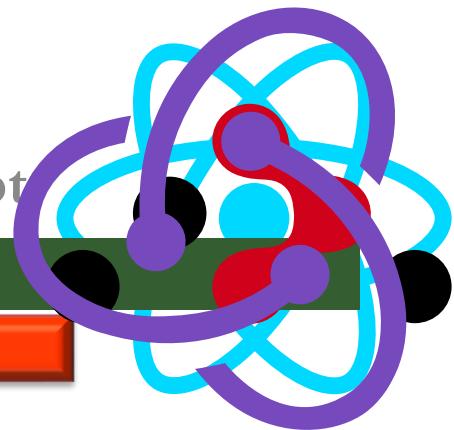
Nous allons définir **BrowserRouter** en haut de **<App />**, afin de nous permettre d'utiliser **<Route>** dans les conteneurs.

```
import React from 'react';
import ReactDOM from 'react-dom';
import {Provider} from 'react-redux'
import {BrowserRouter} from 'react-router-dom'
import store from './store'
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';
ReactDOM.render(
  <Provider store={store}>
    <BrowserRouter>
      <App/>
    </BrowserRouter>
  </Provider> , document.querySelector('#root'));
registerServiceWorker();
```



React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

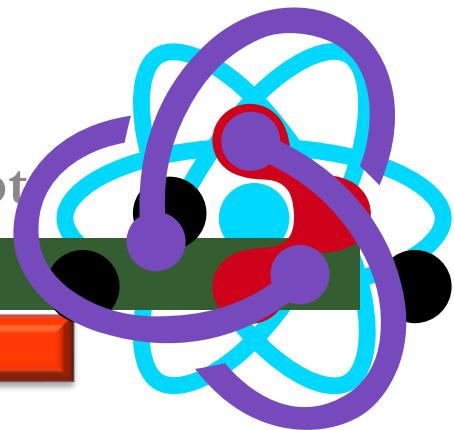
projet

L'organigramme Redux

Commençons par les **containers** puisque les containers sont les principaux gestionnaires de chaque component

```
import {connect} from 'react-redux'  
import Counter from './components/Counter'  
import * as actions from './actions'  
// Map Redux state to component props  
function mapStateToProps(state) {  
    return {  
        count: state.CountReducer.count,  
        wish_value: state.CountReducer.wish_value  
    }  
}  
// Map Redux actions to component props  
function mapDispatchToProps(dispatch) {  
    return {  
        onIncreaseClick: () => dispatch(actions.increaseTodo()),  
        onUpdateClick: event => dispatch(  
            actions.updateTodo(event.target.value)  
        ),  
    }  
}  
// Connected Component  
const VisibleCounter = connect(  
    mapStateToProps,  
    mapDispatchToProps)(Counter);  
export default VisibleCounter;
```





Application monopage

Projet Recapitulatif

projet

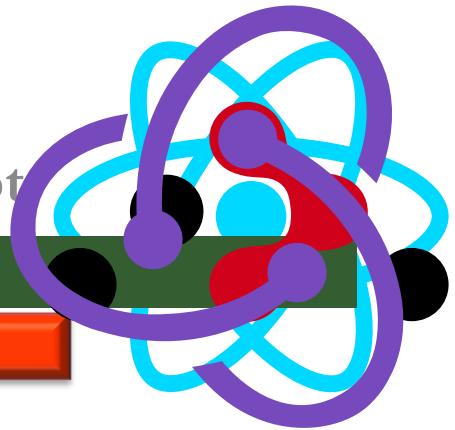
L'organigramme Redux

Il existe deux méthodes principales définies par redux pour chaque containers: **mapStateToProps** et

mapDispatchToProps : parmi lesquelles **mapStateToProps** est chargé de charger les états du magasin et de les mapper dans les accessoires des composants, tandis que **mapDispatchToProps** consiste à envoyer une action aux reducers.

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Composant de conception:

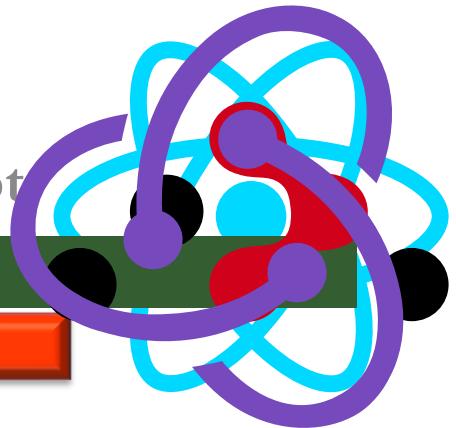
(src / components / Counter.js)

Composant avec champ de saisie et bouton

```
import React, {Component} from 'react'
import PropTypes from 'prop-types'
// React component
export default class Counter extends Component {
  render() {
    const {count, wish_value, onIncreaseClick, onUpdateClick} = this.props
    return (
      <div>
        <span>{count}</span>
        <button onClick={onIncreaseClick}>Increase</button>
        <input value={wish_value} type='text' onChange={onUpdateClick}/>
      </div>
    )
  }
}
```

React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

projet

L'organigramme Redux

Après avoir terminé tous les components, les containers, les reducers, nous avons besoin d'un transporteur qui passe des containers aux reducers et laisse le store changer d'état.

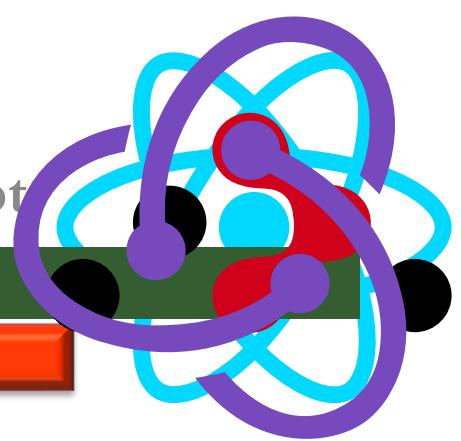
Voici un exemple d'action

```
//Action Creator
export const increaseTodo = () => {
  return {
    type: 'increase',
  }
}
export const updateTodo = (wish_value) => {
  return {
    type: 'update',
    wish_value: wish_value,
  }
}
```



React.JS

un framework JavaScript



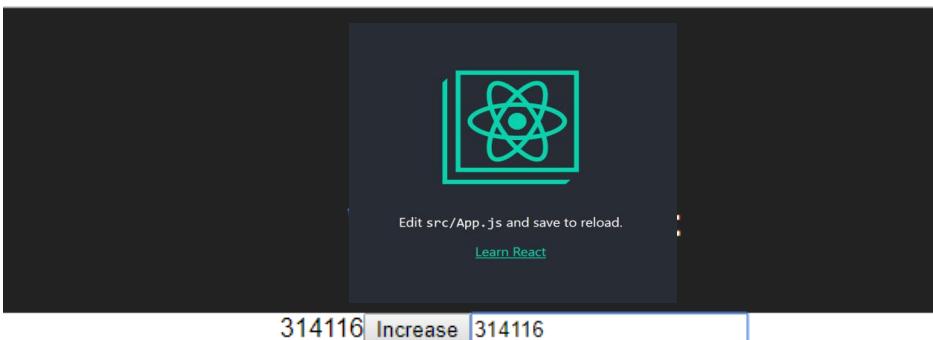
Application monopage

Projet Recapitulatif

projet

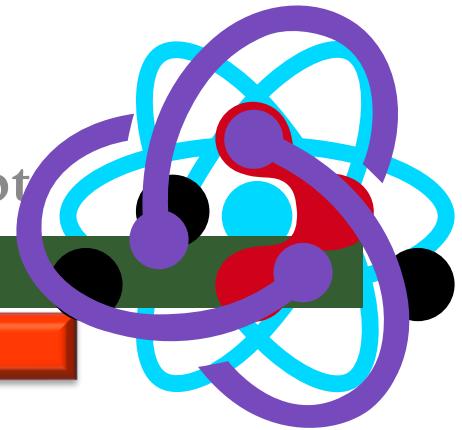
L'organigramme Redux

C:\nodejs\react-projet>npm start



React.JS

un framework JavaScript

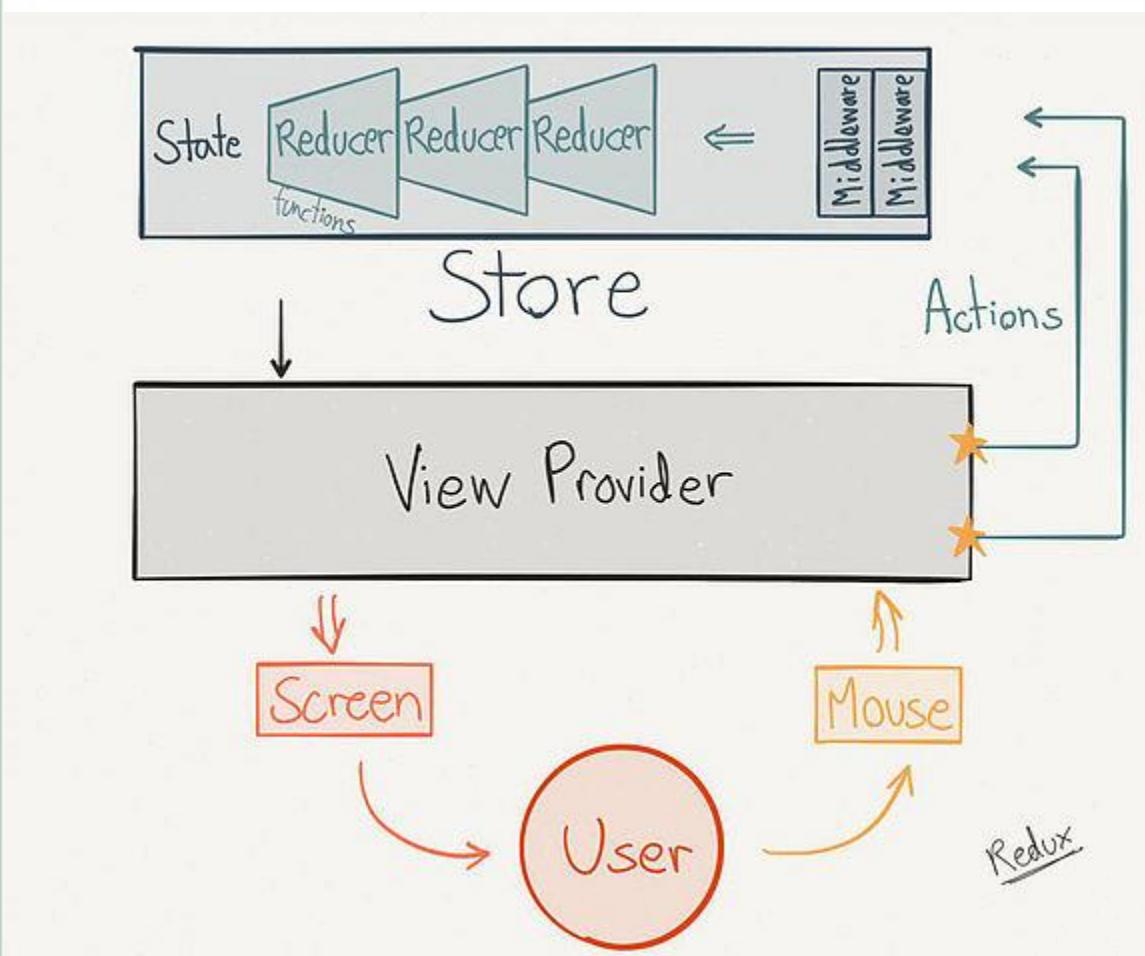


Application monopage

Projet multi reducer

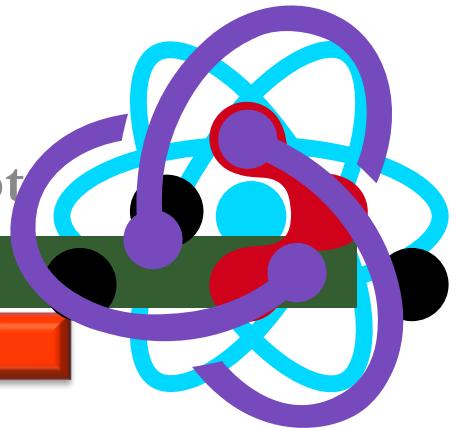
projet

L'organigramme Redux



React.JS

un framework JavaScript



Application monopage

Projet multi reducer

Multi Reducer

Pour le moment nous avons

- Dans le composant nous dispatchons une action
- L'action est transmise au reducer, qui met à jour le state
- Le composant est réaffiché avec la nouvelle valeur du state

Pour qu'une action déclenche un appel http nous allons faire appel à une bibliothèque supplémentaire

Redux-Thunk

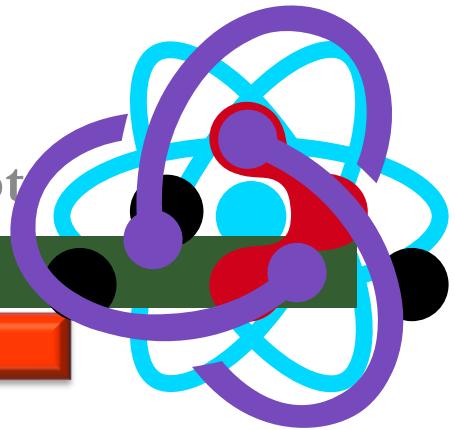
C'est un middleware
pour redux

晦
Redux Thunk-Fu
Fighting with Async



React.JS

un framework JavaScript



Application monopage

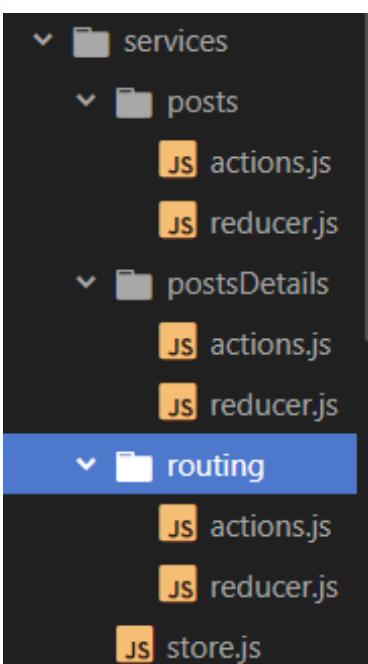
Projet multi reducer

Multi Reducer

Le projet est d'appeler une liste au format JSON

Pour obtenir un détail nous cliquerons sur le titre grâce à permalink de json

Nous allons scinder le store en différents modules nommés "services" (qui contient action et reducer)



Trois services

Posts : récupère la liste des sujets

postDetails ira chercher les détails

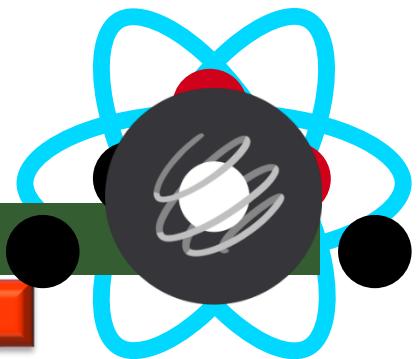
Routing se charge du cycle de vie de l'application (afficher le sujet, afficher la liste une fois cliquer sur Back)

React.JS

un framework JavaScript

Application monopage

FLUX

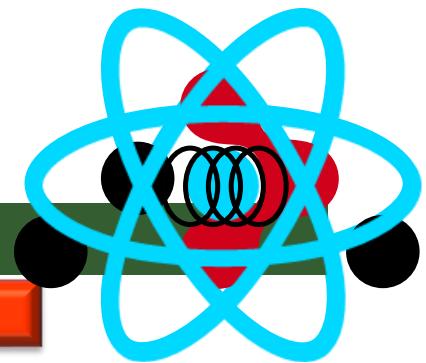


FLUX

Il existe des façons multiples de gérer la gestion de l'État dans React. Essayer de comprendre les options, les compromis entre elles et leur fonctionnement peut être long

Dans ce paragraphe nous allons comparer comment implémenter la gestion globale de l'état dans une application React en utilisant le même modèle dans 5 des bibliothèques et API les plus populaires en utilisant les versions les plus modernes et les plus à jour des bibliothèques.

Recul
MobX
XState
Redux (avec hook)
contexte
Jotaï



Application monopage

Recoil

Recoil.js

RecoilJS est une bibliothèque de gestion d'état pour React qui a été rendue publique récemment par Facebook.

La vérité est qu'ils l'utilisent depuis un certain temps en interne, alors ils ont finalement décidé de l'open source

L'un des problèmes auxquels nous sommes confrontés en matière de gestion de l'État est que les composants enfants doivent constamment informer les ancêtres.

Cela peut sembler assez simple pour les petites applications, mais pour les plus complexes, les choses commencent à devenir laides, en particulier lorsque nous parlons d'arbres de composants longs que nous forçons à obtenir un nouveau rendu à chaque changement déclenché par un enfant imbriqué.

Recoil est une API simple d'utilisation. Elle est assez naturelle pour les développeurs déjà accoutumés aux hooks de React.

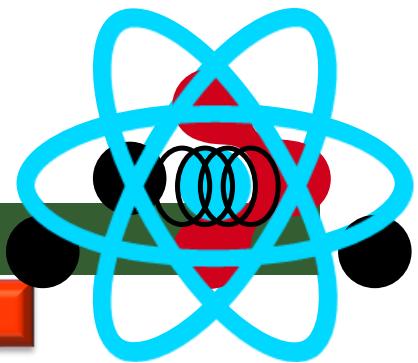
Recoil permet de souscrire exactement aux données que vos composants consomment. On peut aussi créer des selecteurs qui établissent des calculs. La bibliothèque inclue aussi une solution pour effectuer des opérations asynchrones

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

Les Atoms:

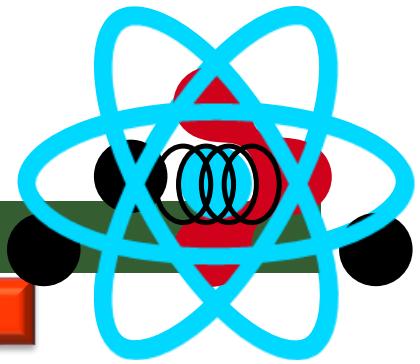
Un Atom est une partie d'un état. Pour imager cela on peut considérer un Atom comme un state React classique que l'on peut accéder dans toute l'application. Changer l'état d'un Atom déclenchera un re render de tous les composants ayant souscrits à ce dernier.

Nous devons lui donner une clé unique ainsi qu'une valeur par défaut. Cette valeur peut être statique, une fonction ou bien une fonction asynchrone.

```
//Déclaration d'un Atom :  
const tempFahrenheitState = atom({  
  key: "tempFahrenheit",  
  default: 0,  
});
```

React.JS

un framework JavaScript



Application monopage

Recoil

Recoil.js

Les Selectors:

Au sens Recoil est un peu différent de ce que l'on connaît. Un selector comme un état dérivé. Il nous permet de construire des données dynamiquement à partir d'autres Atoms. Il possède une fonction obligatoire 'get' (équivalent de reselect avec Redux) mais accepte aussi une fonction optionnelle 'set' permettant de mettre à jour un ou plusieurs Atoms.

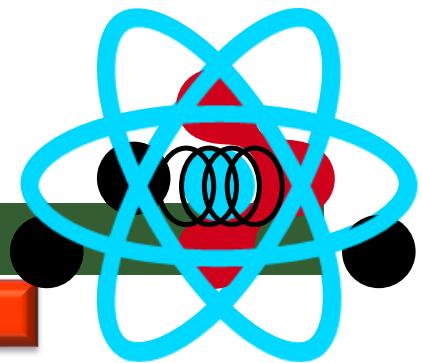
```
const tempCelciusState = selector({
  key: "tempCelcius",
  // la fonction obligatoire "get" pour effectuer le calcul de conversion
  get: ({ get }) => Math.round(((get(tempFahrenheitState) - 32) * 5) / 9)
});
```

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

Le hook useRecoilState:

C'est un hook permettant de souscrire à un Atom en particulier.

Le hook useRecoilValue:

C'est un hook qui retourne uniquement la valeur d'un Atom sans retourner son setter.

Le hook useSetRecoilState:

A contrario, ce hook permet de récupérer le setter d'un Atom sans retourner sa valeur.

Le hook useResetRecoilState:

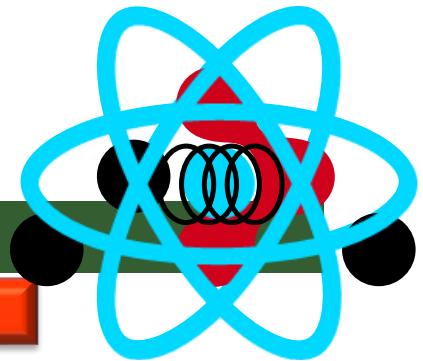
Ce hook permet de remettre à la valeur par défaut un state en nous retournant une fonction de reset.

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

nous parlons principalement React.useContext

pour une gestion des états encore plus flexible sur des applications complexes.

Pour ce faire, il définit un graphique attaché à notre arborescence de composants React afin que les changements d'état découlent d' atoms ce qui sont les racines de ce graphique vers nos composants à travers selectors lesquels sont des fonctions pures.

```
import { atom } from 'recoil';

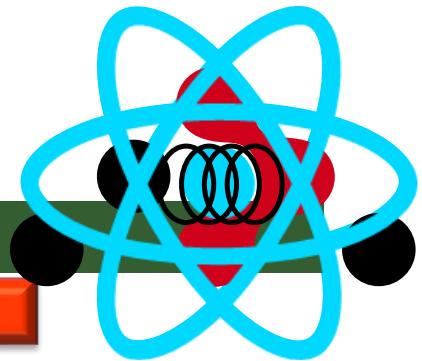
const itemsState = atom({
  key: 'itemsState',
  default: [
    description: 'Don\'t be lazy, write the post of the week 😊',
    done: false,
  ],
});
```

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

ces atoms sont des unités d'État. Chacun atom a un identifiant unique et une valeur par défaut.

Pour nos exemples, nous pouvons utiliser une simple liste de tâches. RecoilJS propose des hook que nous pouvons utiliser pour accéder à l' atomsétat.

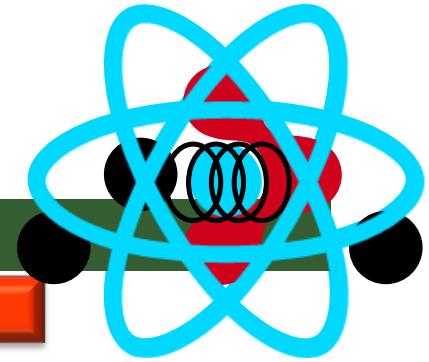
Nous pouvons utiliser un hook nommé useRecoilStatepour accéder à la liste et lui appliquer des modifications ou simplement le hook useRecoilValuepour extraire la liste uniquement.

```
import { atom } from 'recoil';

const itemsState = atom({
  key: 'itemsState',
  default: [
    description: 'Don\'t be lazy, write the post of the week 😊',
    done: false,
  ],
});
```

React.JS

un framework JavaScript



Application monopage

Recoil

Recoil.js

les sélecteurs sont de simples fonctions et nous les utilisons pour extraire des données d' atoms ou même d'autres selectors .

Ils peuvent obtenir ces deux éléments en entrée et ils sont réévalués à chaque changement d'état.

Les composants qui s'y abonnent, sont rendus à chaque fois en conséquence.

```
const itemsState = atom({
  key: 'itemsState',
  default: [
    {
      description: 'Don\'t be lazy, write the post of the week 😊',
      done: false,
    },
  ],
});

const unfinishedItemsState = selector({
  key: 'unfinishedItemsState',
  get: ({ get }) => {
    const items = get(itemsState);

    return items.filter(item => item.done === false);
  }
});

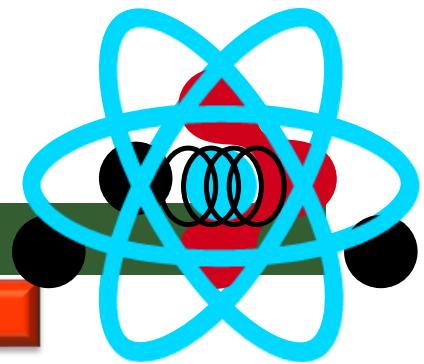
const unfinishedItemsCountState = selector({
  key: 'unfinishedItemsCountState',
  get: ({ get }) => {
    const items = get(unfinishedItemsState);

    return items.length;
  }
});
```



React.JS

un framework JavaScript



Application monopage

Recoil

Recoil.js

Nous avons ajouté une nouvelle constante `unfinishedItemsCountState` qui extrait les éléments inachevés du **unfinishedItemsStatesélecteur** et renvoie leur longueur. Utilisons ensuite **unfinishedItemsCountState** dans notre composant

```
const itemsState = atom({
  key: 'itemsState',
  default: [
    {
      description: 'Don\'t be lazy, write the post of the week 😊',
      done: false,
    },
  ],
});

const unfinishedItemsState = selector({
  key: 'unfinishedItemsState',
  get: ({ get }) => {
    const items = get(itemsState);

    return items.filter(item => item.done === false);
  }
});

const unfinishedItemsCountState = selector({
  key: 'unfinishedItemsCountState',
  get: ({ get }) => {
    const items = get(unfinishedItemsState);

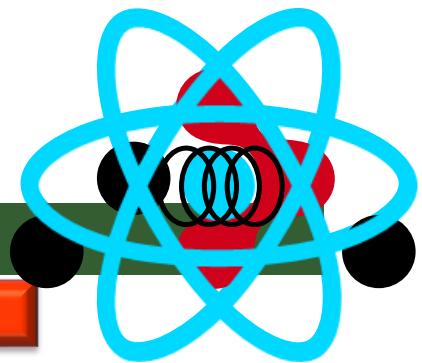
    return items.length;
  }
});
```

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

Comme nous l'avons vu ci-dessus, useRecoilValuehook renvoie la liste des éléments elle-même.

Pour mettre à jour cette liste, nous pourrions utiliser un useRecoilStatecrochet qui expose une méthode que nous pouvons utiliser pour mettre à jour la liste en conséquence. Il est temps de créer une entrée contrôlée à l'aide de React.useState, puis de profiter du useRecoilState hook pour ajouter de nouveaux éléments à notre liste :

```
const List = () => {
  const unfinishedItemsCount = useRecoilValue(unfinishedItemsCountSt
  const [items, setItems] = useRecoilState(itemsState);
  const [value, setValue] = React.useState('');

  const handleSubmit = e => {
    e.preventDefault();
    setItems(items.concat({
      description: value,
      done: false,
    }));
    setValue('');
  };

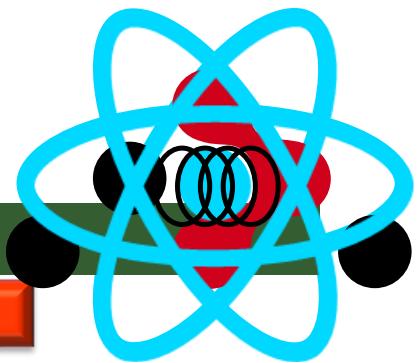
  return (
    <>
      You have {unfinishedItemsCount} unfinished tasks!!
    </>
  );
}
```

React.JS

un framework JavaScript

Application monopage

Recoil



Recoil.js

RecoilJS utilise le contexte pour créer ces liaisons exceptionnelles, c'est pourquoi nous devons envelopper notre Appcomposant de niveau supérieur avec RecoilRoot.

Il s'agit d'un fournisseur de contexte fourni par RecoilJS. Set doit être l'ancêtre de tous les composants qui utilisent atomset selectors

```
import React from 'react';
import ReactDOM from 'react-dom';
import { RecoilRoot } from 'recoil';

import App from './App';

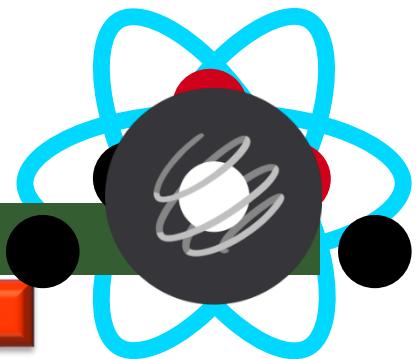
ReactDOM.render(
  <RecoilRoot>
    <App />
  </RecoilRoot>,
  document.getElementById('root'),
);
```

React.JS

un framework JavaScript

Application monopage

Jōtai



Jōtai

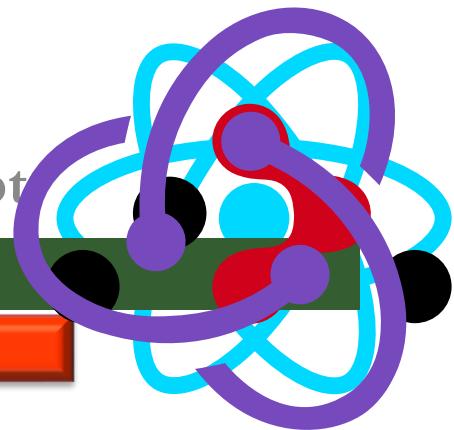
Nouveau state manager pour React (state en japonais), inspiré de Recoil, avec une api simplifiée. Sous la gouverne de React-Spring, qui va bientôt se rebrander sous forme de collectif.

```
import { atom } from 'jotai'  
const countAtom = atom(0)  
const countryAtom = atom("Japan")  
const citiesAtom = atom(["Tokyo", "Kyoto", "Osaka"])  
const mangaAtom = atom({ "Dragon Ball": 1984, "One Piece": 1997, "Naruto": 1999 })
```



React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

Importer une production

Product One Product Two Product Three



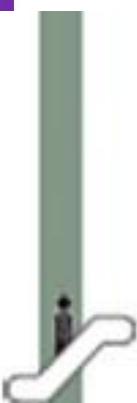
Small **Medium** Large

Product One

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

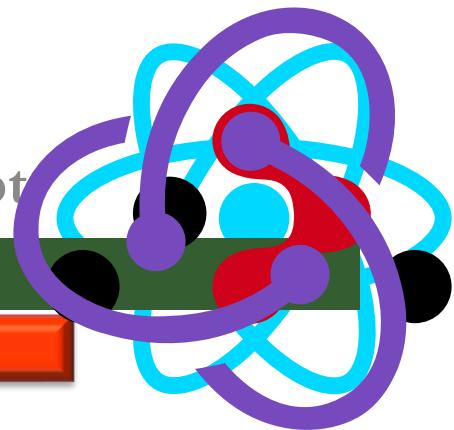
\$19.99

Add to Cart



React.JS

un framework JavaScript



Application monopage

Projet Recapitulatif

Importer une production

A partir d'un projet déjà développé il est possible de copier-coller
Les dossiers src, public et le fichier package.json
Et de lancer une mise à jour des dépendances

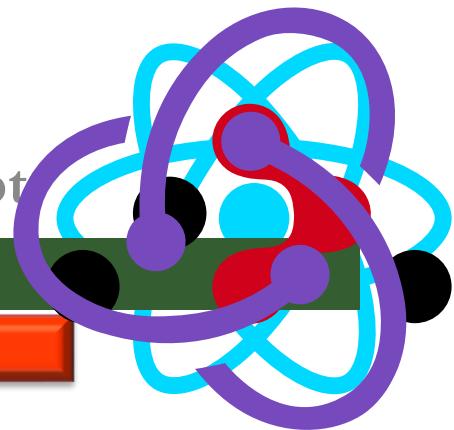
C:\nodejs\react-anim>npm install

```
"name": "react-animation-example",
"version": "0.1.0",
"private": true,
"dependencies": {
  "react": "^16.8.2",
  "react-animation": "^1.0.9",
  "react-dom": "^16.8.2",
  "react-router-dom": "^4.3.1",
  "react-scripts": "2.1.5",
  "styled-components": "^4.1.3"
},
```



React.JS

un framework JavaScript



Application monopage

Avec context

Context

Depuis fin mars 2018, la version 16.3 de React est sortie et a apporté son lot de nouveautés, dont l'API Context.

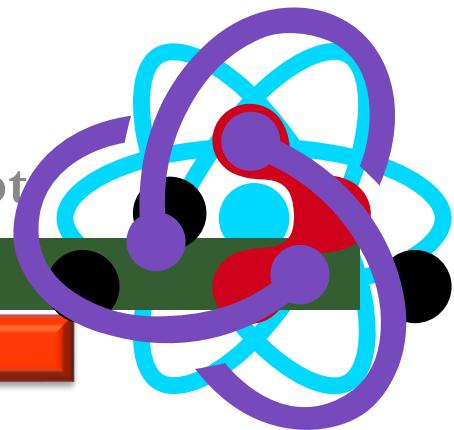
Context permet de rendre disponibles des propriétés au sein des ses composants React sans avoir à les passer directement à ces derniers.

Lorsqu'on a une application un peu complexe (entendre beaucoup de composants et d'héritages de propriétés), il devient très vite compliqué de maintenir tout ces états.

Avec la nouvelle API Context, on peut créer un ou plusieurs stores pour nos données.

React.JS

un framework JavaScript



Application monopage

Avec context

Context

l'API, on a deux propriétés :

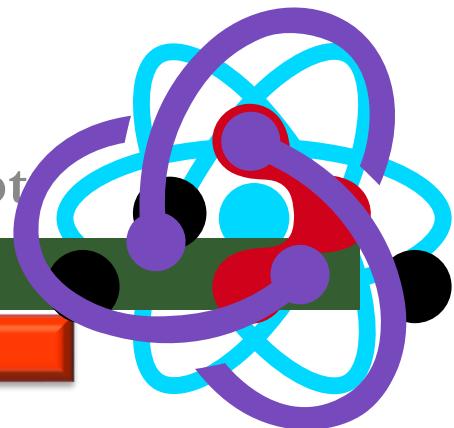
le Provider, qui se charge de diffuser nos propriétés d'une part,
et un ou plusieurs **Consumers** qui permettent d'accéder aux données
fournies par le Provider d'autre part.

```
import React, { createContext, Component } from "react";
export const UserContext = createContext({
  name: "",
  setName: () => {}
});

class UserProvider extends Component {
  state = {"Formation react", // une valeur de départ
    setName: name => this.setState({ name: name })
  };
  render() {
    return (
      <UserContext.Provider value={this.state}>
        {this.props.children}
      </UserContext.Provider>
    );
  }
}

export const withUser = Component => props => (
  <UserContext.Consumer>
    {store => <Component {...props} {...store} />}
  </UserContext.Consumer>;
)

export default UserProvider;
```



Application monopage

Avec context

Context

Avec l'API Context, on peut :

- créer des "micro stores" pour certaines parties de notre application, et les faire hériter d'un store plus global.
- imaginer combiner les stores et les faire "hériter" les uns des autres.

On résout au passage pas mal de problèmes liés à l'imbrication et à la hiérarchisation des composants.

Ainsi, on peut très facilement faire face à une application qui grossit sans avoir à utiliser Redux plus lourds à mettre en place

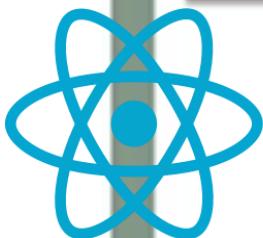
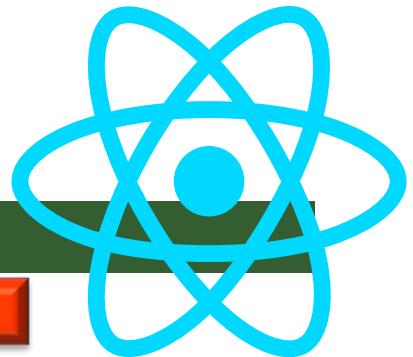
React.JS

un framework JavaScript

- Rappels des composants des RIA
- Développer avec ReactJS
- Interactivité des composants
- Application monopage avec ReactJS Redux
- Application isomorphique
- Introduction à React Native

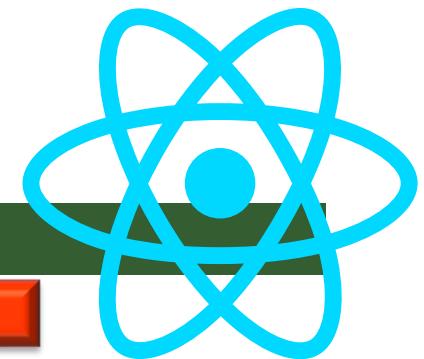
A large green word cloud centered on the page, containing terms related to React.js development such as 'composants', 'react', 'jsx', 'redux', 'propriétés', 'patterns', 'application', 'écosystème', 'gestion', 'contrôle', and 'native'.

- Positionnement, différences avec Cordova.
- De React aux composant iOS natifs



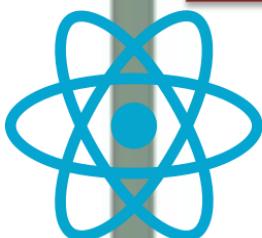
React Native

React Native, vous permettra d'utiliser, des composants natifs des OS mobiles cités, en écrivant du javascript. Lorsque nous écrirons notre première application mobile – en javascript, un “Hello World” – donc afficher juste un “Hello World” dans une balise “View” (balise d'affichage d'élément(s) de **React Native**, à l'instar du div en HTML), ce framework saura la traduire en UIView pour iOS et en android.view.View pour Android.



React Native

Cordova et autres



React Native

Développement mobile

Les applications cross-platforms, à l'inverse des applications natives, sont **développées une seule et unique fois** et sont **compatibles sur iOS et Android**.

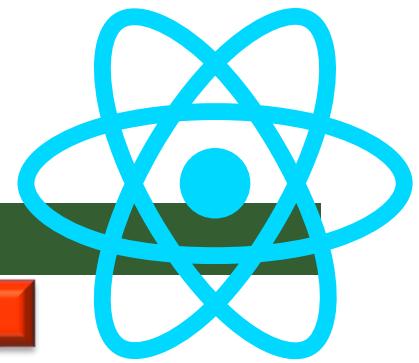
Le développement d'applications cross-platforms passe par des frameworks. Parmi les plus connus, on retrouve Ionic, PhoneGap, Xamarin et Titanium

Plus besoin d'apprendre un langage de programmation par plateforme.

On développe notre application une fois, dans un langage et le framework se charge de vous créer une application compatible iOS et Android.

React.JS

un framework JavaScript



React Native

Cordova et autres

Développement mobile

lorsque vous définissez par exemple une vue en React Native,
sur iOS votre application affiche

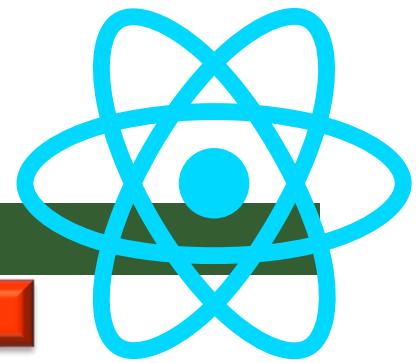
une **UIView** et sur Android une
android.view.View, deux composants natifs.

Le fonctionnement est le même pour tous les types d'éléments
graphiques : boutons, textes, listes, chargement, etc. React Native
convertit tous vos éléments en leur équivalent natif



React.JS

un framework JavaScript



React Native

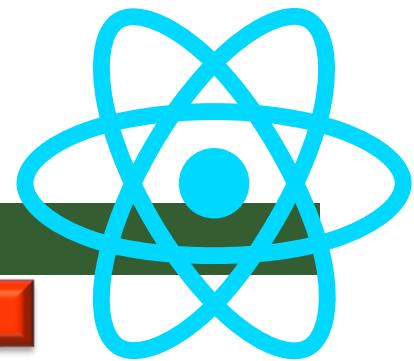
Cordova et autres

Développement mobile

lorsque vous développez une application mobile, vous avez parfois besoin d'accéder à des **composants du téléphone**, par exemple la caméra, la géolocalisation, les contacts, etc. Dès lors que vous souhaitez utiliser un composant du téléphone, il faut utiliser du **code natif**

React.JS

un framework JavaScript



React Native

Cordova et autres

<https://facebook.github.io/react-native/docs/getting-started>

The Basics

- Getting Started
- Learn the Basics
- Props
- State
- Style
- Height and Width
- Layout with Flexbox
- Handling Text Input
- Handling Touches
- Using a ScrollView
- Using List Views
- Networking
- More Resources

Guides

- Components and APIs
- Platform Specific Code
- Navigating Between Screens

Getting Started

This page will help you install and build your first React Native app. If you can skip ahead to the [Tutorial](#).

If you are coming from a web background, the easiest way to get started because they allow you to start a project without installing and configuring a development environment on your local machine and you can build it in minutes. For instant development, you can use [Snack](#) to try React Native code.

If you are familiar with native development, you will likely want to use Android Studio to get started. If you already have one of these tools installed and running within a few minutes. If they are not installed, you should expect to spend time installing and configuring them.

[Expo CLI Quickstart](#) [React Native CLI Quickstart](#)

Assuming that you have [Node 10+](#) installed, you can use npm to install the Expo CLI:

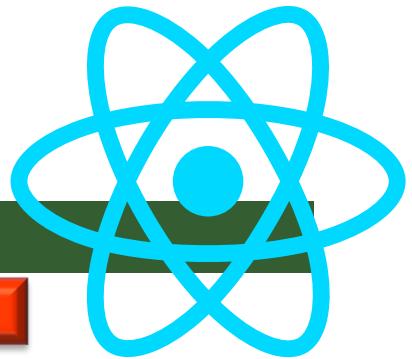
```
npm install -g expo-cli
```

React.JS

un framework JavaScript

React Native

Cordova et autres



Développement mobile

create-react-native-app

Dans le premier onglet, *Quick Start*, React Native vous explique comment créer une `create-react-native-app`.

Nous allons utiliser cette solution **CRNA** (**Create-React-Native-App**)

grâce à Node.js, vous pouvez installer les lignes de commande CRNA.
Ces lignes de commande nous permettront, un peu plus loin,
de **créer une CRNA**

`C:\nodejs\npm install -g react-native-app`

`C:\nodejs\npm install -g react-native-cli`

Ou

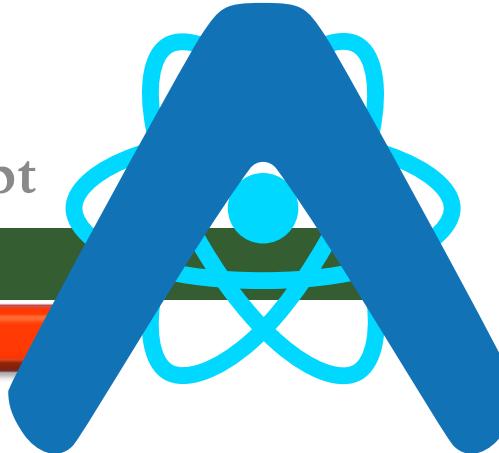
`C:\nodejs\npm init react-app my-app`

React.JS

un framework JavaScript

React Native

Cordova et autres



Expo

En plus de nous procurer un outil de développement d'application React Native, Expo nous offre une application hôte.

Cette application va nous permettre d'exécuter et d'afficher le rendu de nos applications pendant les phases de développement.

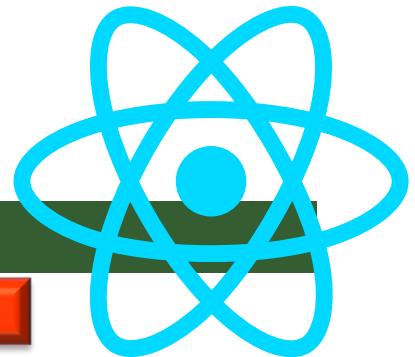
Si vous possédez un smartphone / une tablette iOS ou Android

Vous pouvez télécharger l'application Expo.

L'application est disponible sur l'App Store et sur le Google Play.

React.JS

un framework JavaScript



React Native

Cordova et autres

Lancement du projet

```
C:\nodejs\npm install -g expo-cli
```

```
C:\nodejs\react-native> expo init test-natif
```

```
C:\nodejs>cd react-native

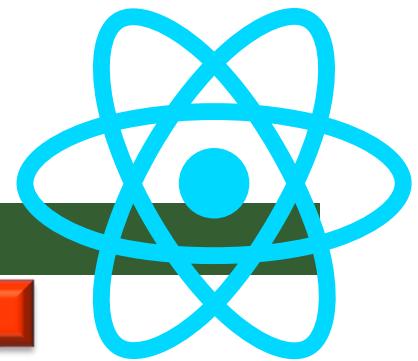
C:\nodejs\react-native> expo init MoviesAndMe
? Choose a template: expo-template-blank
? Please enter a few initial configuration values.
  Read more: https://docs.expo.io/versions/latest/workflow/configuration/ » 50% completed
{
  "expo": {
    "name": "<The name of your app visible on the home screen>",
    "slug": "MoviesAndMe"
  }
}
```

Lancement du projet dans le répertoire

```
C:\nodejs\react-native> expo start
```

React.JS

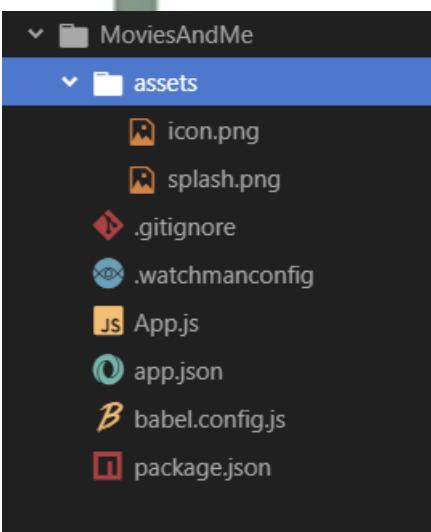
un framework JavaScript



React Native

Cordova et autres

Lancement du projet



babel.config.js

.gitignore

.watchmanconfig

assets

Babel est un compilateur Javascript. Il permet d'écrire du Javascript avec une syntaxe plus simple, plus moderne, tout en restant compatible avec les normes Javascript plus ou moins anciennes. Le fichier `babel.config.js` correspond à sa configuration. Ici, on utilise la configuration d'expo qui correspond à la dernière en date (ES6) et qui offre les dernières fonctionnalités de développement.

Pour ceux qui utilisent **Git**, un fichier `gitignore` est déjà configuré pour exclure les fichiers React Native et Expo de vos repos.

Watchman est un service permettant d'observer notre projet. Par défaut, Watchman regarde et notifie chaque changement dans votre projet. Ce fichier de configuration permet de modifier ce comportement en excluant des fichiers de ce service.

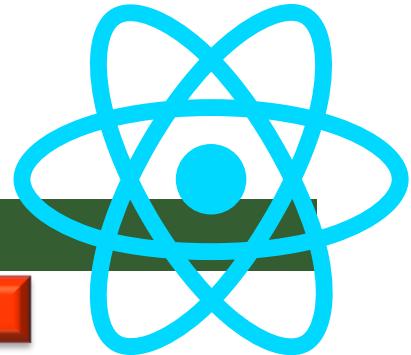
Ce dossier contient deux images. L'une correspond à l'**icône de votre application** et l'autre à son **splashscreen**, il s'agit de l'image affichée lorsque l'application est lancée. Expo vous permet ici de modifier ces deux images afin qu'elles représentent au mieux votre application.

React.JS

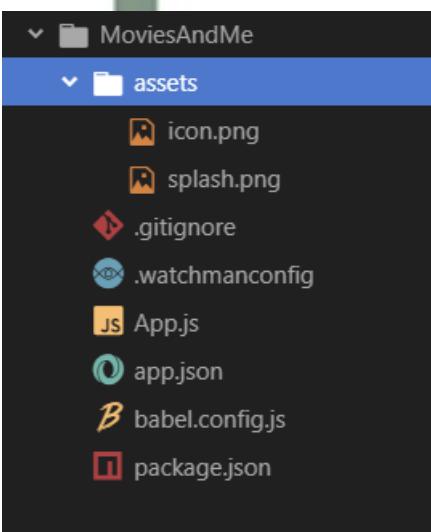
un framework JavaScript

React Native

Cordova et autres



Lancement du projet



node_modules

En parlant de fichier, on commence par un dossier. Si vous dépliez ce dossier, vous devez voir une multitude de dossiers.

app.json

Ce fichier permet de configurer des informations liées à votre application. On définit le SDK Expo.

point d'entrée de votre CRNA

C'est ici que sont recensées toutes les dépendances de votre application.

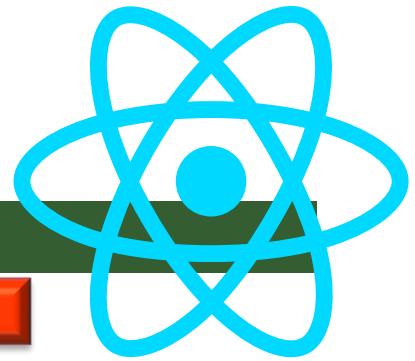
Si ce fichier est bien configuré, une personne qui reprend votre travail n'aura qu'une commande à taper avant de se lancer dans les développements.

Si vous ouvrez ce fichier, vous remarquerez qu'on définit également le nom de votre application (celui qui s'affichera sur Expo) et la version de votre application.

package.json

React.JS

un framework JavaScript



React Native

Cordova et autres

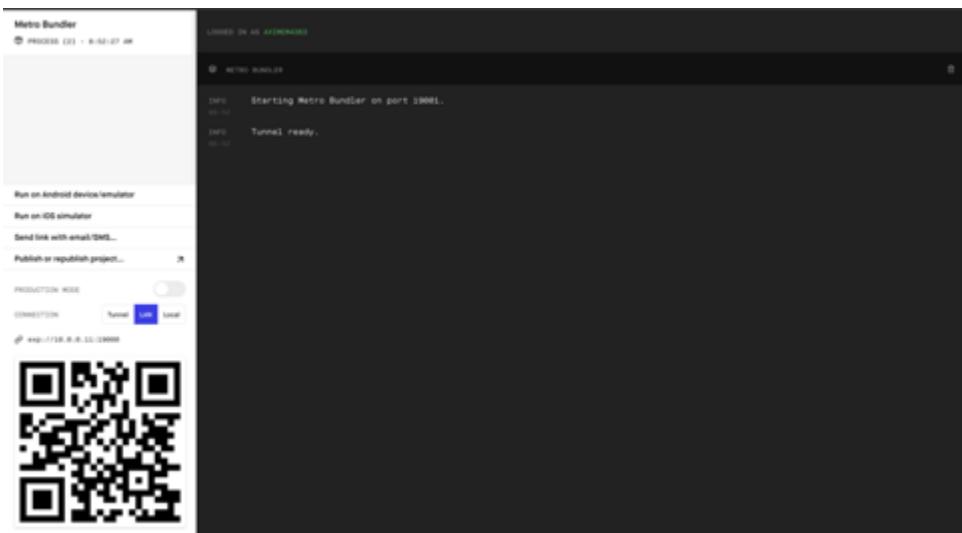
Lancement du projet

C:\nodejs>cd MoviesAndMe

Si erreur avec node 12

<https://github.com/expo/expo-cli/issues/1074>

C:\nodejs\react-native\MoviesAndMe> npm start

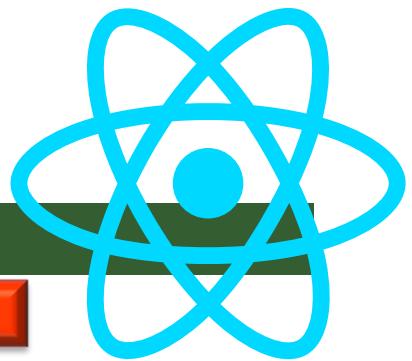


React.JS

un framework JavaScript

React Native

Cordova et autres



Lancement du projet

Créer un dossier Components à la racine de notre projet et à créer un fichier Search.js à l'intérieur de ce dossier.

Dans Search.js, on va créer un component custom Search qui va correspondre à notre vue Recherche.

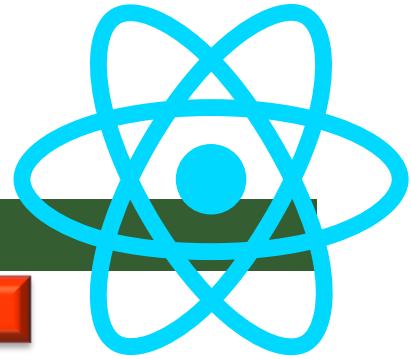
Pour créer un component custom, on a besoin d'importer la librairie React. Rappelez-vous le React.Component de notre fichier App.js.
Comportement/Search.js

```
import React from 'react'
import { View, TextInput, Button } from 'react-native'

class Search extends React.Component {
  render() {
    return (
      <View>
        <TextInput placeholder='Titre du film' />
        <Button title='Rechercher' onPress={() => {}}/>
      </View>
    )
  }
}
export default Search
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Lancement du projet

À présent, on importe notre component dans notre fichier App.js, le point d'entrée de notre application, et on l'affiche comme toute première vue de l'application.

App.js

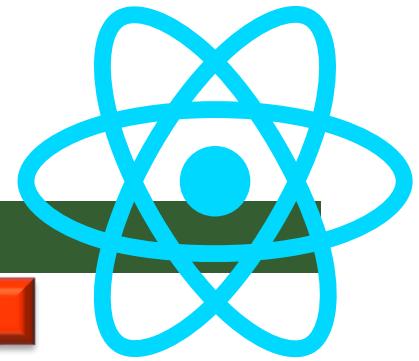
```
import React from 'react'
import { StyleSheet, View, TextInput, Button, Text } from 'react-native'

class Search extends React.Component {
  render() {
    return (
      <View style={styles.main_container}>
        <TextInput style={styles.textinput} placeholder="Titre du
        film"/>
        <Button title='Rechercher' onPress={() => {}}/>
      </View>
    )
  }
}

...
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Lancement du projet

À présent, on importe notre component dans notre fichier App.js, le point d'entrée de notre application, et on l'affiche comme toute première vue de l'application.

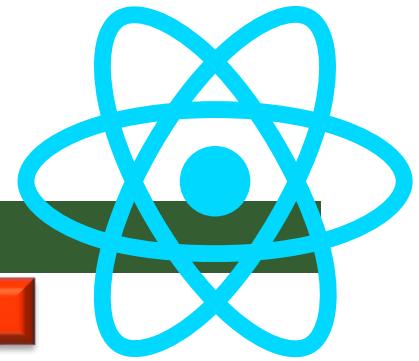
App.js

```
...
const styles = StyleSheet.create({
  main_container: {
    flex: 1,
    marginTop: 20
  },
  textinput: {
    marginLeft: 5,
    marginRight: 5,
    height: 50,
    borderColor: '#000000',
    borderWidth: 1,
    paddingLeft: 5
  }
})
```

```
export default Search
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Lancement du projet

À présent, on importe notre component dans notre fichier App.js, le point d'entrée de notre application, et on l'affiche comme toute première vue de l'application.

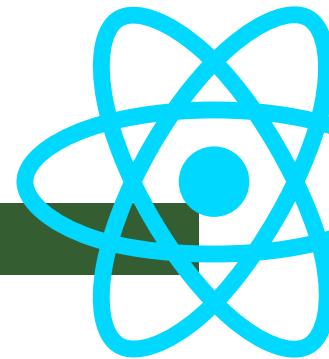
App.js

```
...
const styles = StyleSheet.create({
  main_container: {
    flex: 1,
    marginTop: 20
  },
  textinput: {
    marginLeft: 5,
    marginRight: 5,
    height: 50,
    borderColor: '#000000',
    borderWidth: 1,
    paddingLeft: 5
  }
})
```

```
export default Search
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Autre possibilité en ligne <https://snack.expo.io/>

Modifier le fichier

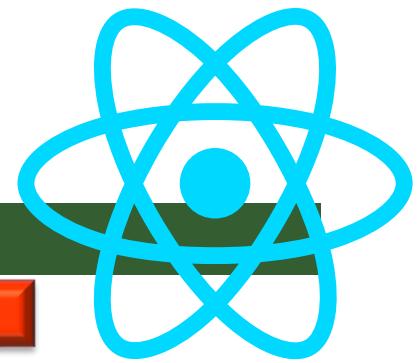
App.js

```
import React from 'react';
import Inputs from './inputs.js'

const App = () => {
  return (
    <Inputs />
  )
}
export default App
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Autre possibilité en ligne <https://snack.expo.io/>

Créer le fichier

inputs.js

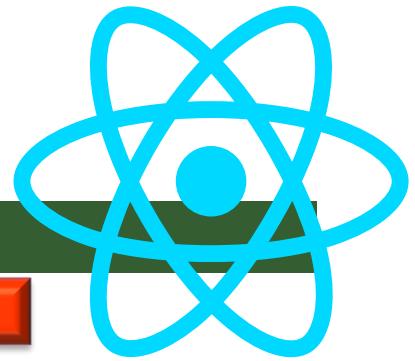
```
import React, { Component } from 'react'
import { View, Text, TouchableOpacity, TextInput, StyleSheet } from
'react-native'

class Inputs extends Component {
  state = {
    email: "", password: ""
  }
  handleEmail = (text) => {
    this.setState({ email: text })
  }
  handlePassword = (text) => {
    this.setState({ password: text })
  }
  login = (email, pass) => {
    alert('email: ' + email + ' password: ' + pass)
  }
  render() {
    return (
      <View style = {styles.container}>
        <TextInput style = {styles.input}
          underlineColorAndroid = "transparent"
          placeholder = "Email"
          placeholderTextColor = "#9a73ef"
          autoCapitalize = "none"
          onChangeText = {this.handleEmail}>
        ...
      </View>
    )
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center"
  },
  input: {
    width: 250,
    height: 40
  }
})
```

React.JS

un framework JavaScript



React Native

Cordova et autres

Autre possibilité en ligne <https://snack.expo.io/>

Créer le fichier
inputs.js

...

```
<TextInput style = {styles.input}
underlineColorAndroid = "transparent"
placeholder = "Password"
placeholderTextColor = "#9a73ef"
autoCapitalize = "none"
onChangeText = {this.handlePassword}/>
<TouchableOpacity style = {styles.submitButton} onPress = { () =>
this.login(this.state.email, this.state.password) }>
<Text style = {styles.submitButtonText}> Submit </Text>
</TouchableOpacity>
</View> )
} }
export default Inputs
const styles = StyleSheet.create({
container: { paddingTop: 23 },
input: { margin: 15, height: 40, borderColor: '#7a42f4', borderWidth: 1 },
submitButton: { backgroundColor: '#7a42f4', padding: 10, margin: 15,
height: 40, },
submitButtonText:{ color: 'white' }
})
```



React.JS

un framework JavaScript

Merci

Si vous avez la moindre question
s.brunet@leserveur.com

