

# Orange Digital Center

## React JS

Maîtriser le framework JavaScript de Facebook



ENICarthage - Les 22, 23 et 24 novembre 2021



Orange  
Digital Center



# Formateur

---

Anis ASSAS

- Ingénieur en informatique
- Enseignant universitaire (M. Technologue à l'ISET Djerba)
- Formateur & conseiller expert

Email : [assas\\_anis@yahoo.fr](mailto:assas_anis@yahoo.fr)

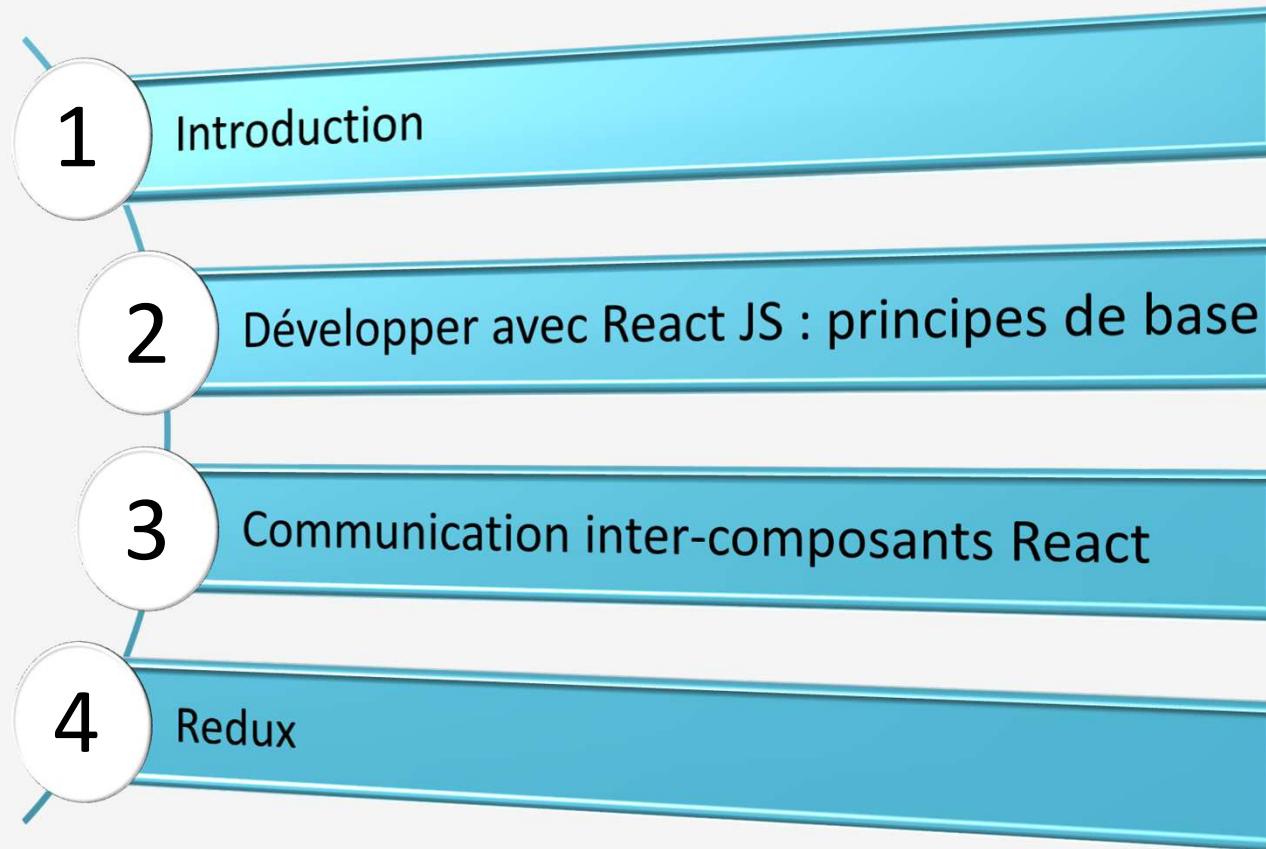
# Fiche formation

---

- **Durée :** 3 jours
- **Prérequis :**
  - Concepts de base de la programmation : variables, opérateurs, fonctions, classes/objets, tableau, ...
  - Fondements de base de développement web : HTML, CSS, JavaScript
- **Objectifs / Compétences visées :**
  - Connaître les fondements de React
  - Développer une application cliente avec ReactJS
  - Création d'une application SPA avec React et Redux.
- **Méthodes pédagogiques :**
  - Notions théoriques et notes de cours
  - Démonstrations et activités pratiques

# Plan de la formation

---



# Planning de la formation

Jour 1

Introduction

**Atelier 1 :**  
Mise en place de l'environnement  
du travail & Initiation JSX

Développer avec  
ReactJS

**Atelier 2 :**  
Création de simples composants  
React

Jour 2

Communication  
inter-composants

**Atelier 3 :**  
Création d'un ensemble structuré  
de composants React

Redux

**Atelier 4 :**  
Dév. application SPA avec React  
& Redux

Jour 3

Evaluation &  
clôture

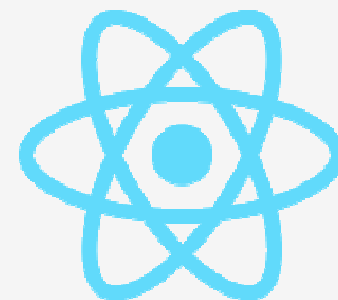


# Introduction

# Introduction : Plan

---

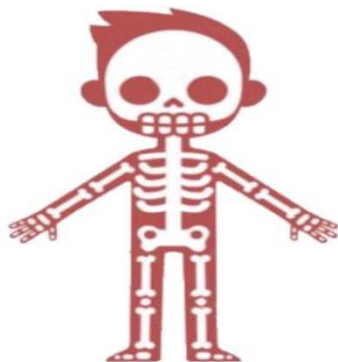
- ❑ Technologies web
- ❑ Les frameworks JavaScript
- ❑ Rappels : JavaScript ES2015 (ES6)
- ❑ Initiation à JSX
- ❑ Présentation générale du ReactJS
- ❑ Installation de l'environnement du travail



🌟 **Atelier** : Mise en place de l'environnement de développement & Initiation à JSX.

# Technologies web

- **HTML : HyperText Markup Language** (langage de balisage hypertexte ou **HTML**) est le langage utilisé pour décrire et définir le contenu d'une page web.
- **CSS : Cascading Style Sheets** (feuilles de style en cascade ou **CSS**) est utilisé pour décrire l'apparence du contenu d'une page web.
- **JavaScript**: c'est un langage de programmation de scripts principalement employé dans les pages web interactives (mais aussi pour les serveurs avec l'utilisation de **Node.js**).



**HTML**



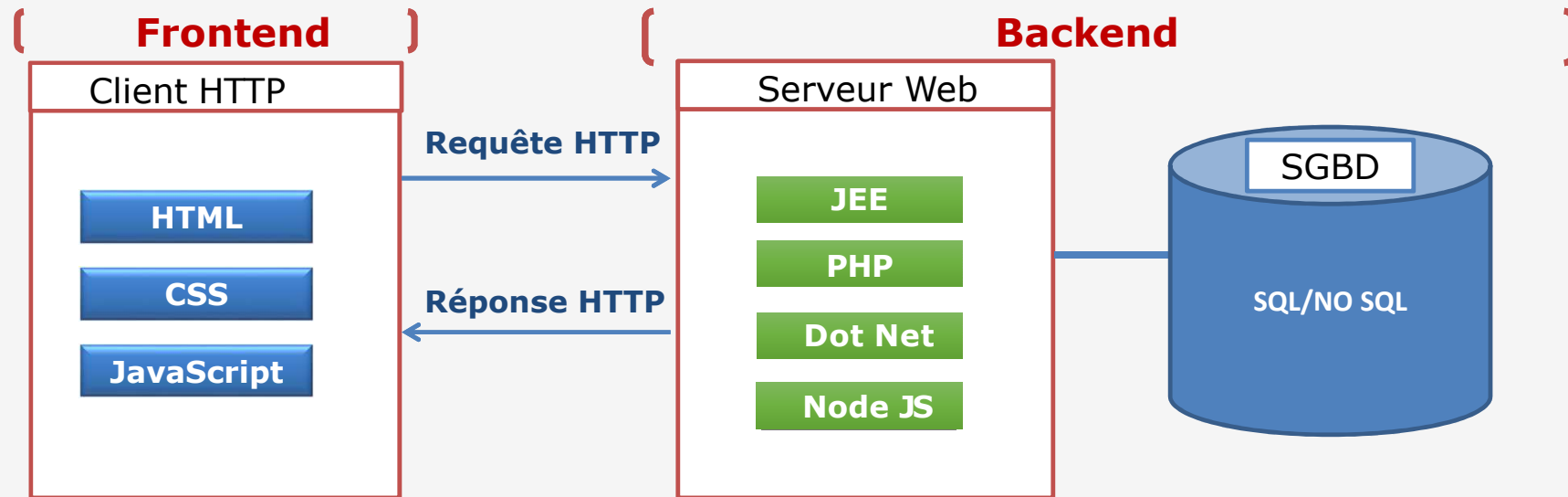
**JavaScript**



**CSS**



# Technologies Web

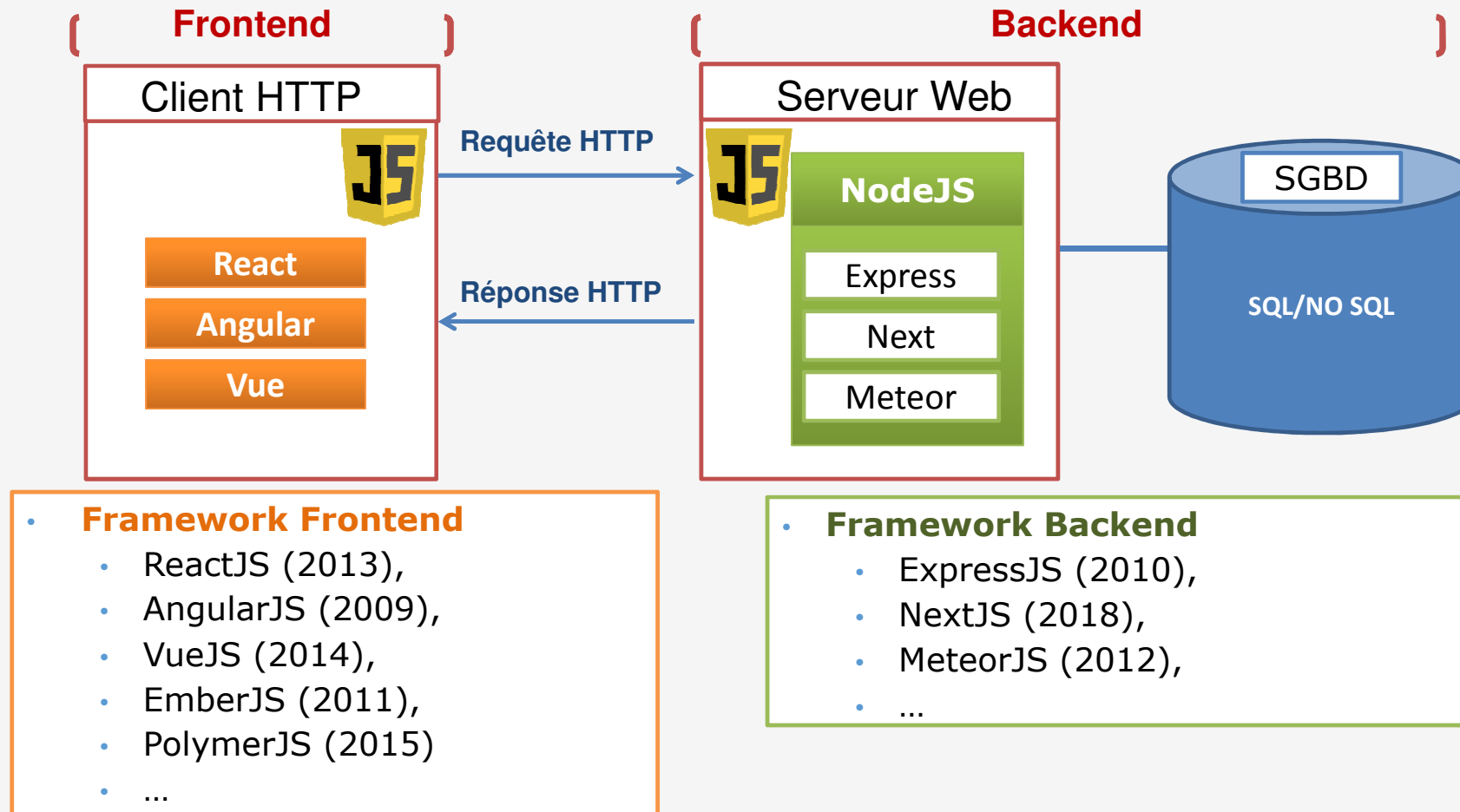


- Un client web (Browser) communique avec le serveur web en utilisant le protocole **HTTP**
- Une application web se compose de deux parties:
  - La partie **Frontend** : S'occupe de la présentation des IHM côté Client :
    - Langages utilisés : HTML, CSS, Java Script
  - La partie **Backend** : S'occupe des traitements effectués côté serveur :
    - Technologies utilisées : PHP, JEE, .Net, Node JS
- La communication entre la partie **Frontend** et la partie **Backend** se fait en utilisant le protocole **HTTP**

# Technologies web

- **Notion de framework :**
  - Plusieurs traductions : cadriceil, environnement de développement, cadre d'applications, ...
  - Ensemble de composants logiciels
  - Facilitant la réalisation d'une (partie de l') application
  - Imposant une certaine structure, logique, syntaxe...
- **Frameworks applicatifs pour le développement d'applications web :**
  - **Angular, React** pour JavaScript,
  - **Spring** pour Java,
  - **Symfony** pour PHP, ...
- **Frameworks de présentation de contenu web : Bootstrap** pour CSS, ...
- **Frameworks de persistance de données : hibernate, ...**
- **Frameworks de logging : log4j, ...**
- ...

# Frameworks JavaScript



# Les frameworks javaScript : évolution

- Evolution JavaScript :



# Frameworks javaScript : utilisation

JavaScript Developer Survey - Edition 2020 (<https://2020.stateofjs.com/fr-FR/technologies/front-end-frameworks/>)

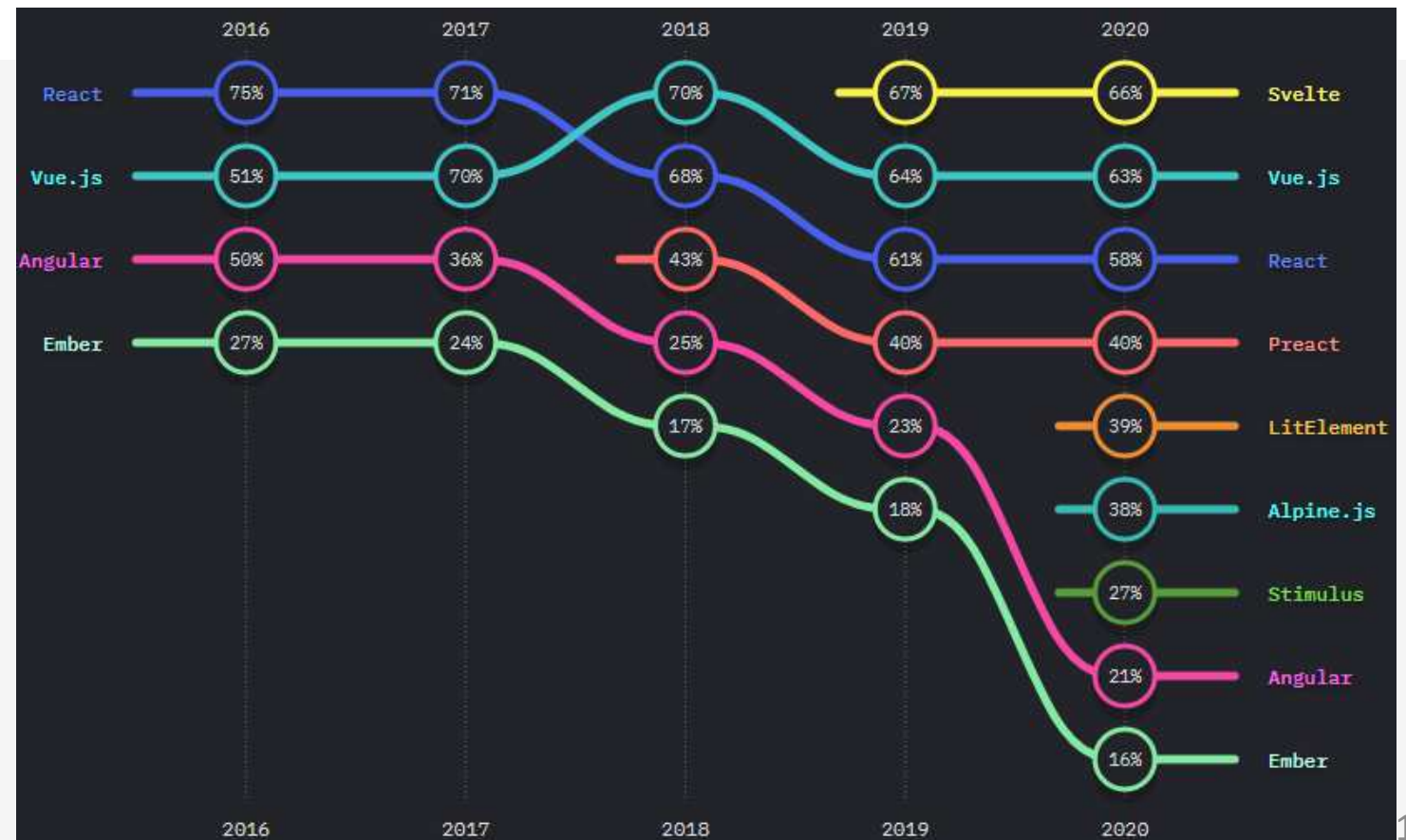
Classement des frameworks Front-End les plus populaires suivant les ratios d'utilisation



# Frameworks javaScript : intérêt

JavaScript Developer Survey - Edition 2020 (<https://2020.stateofjs.com/fr-FR/technologies/front-end-frameworks/>)

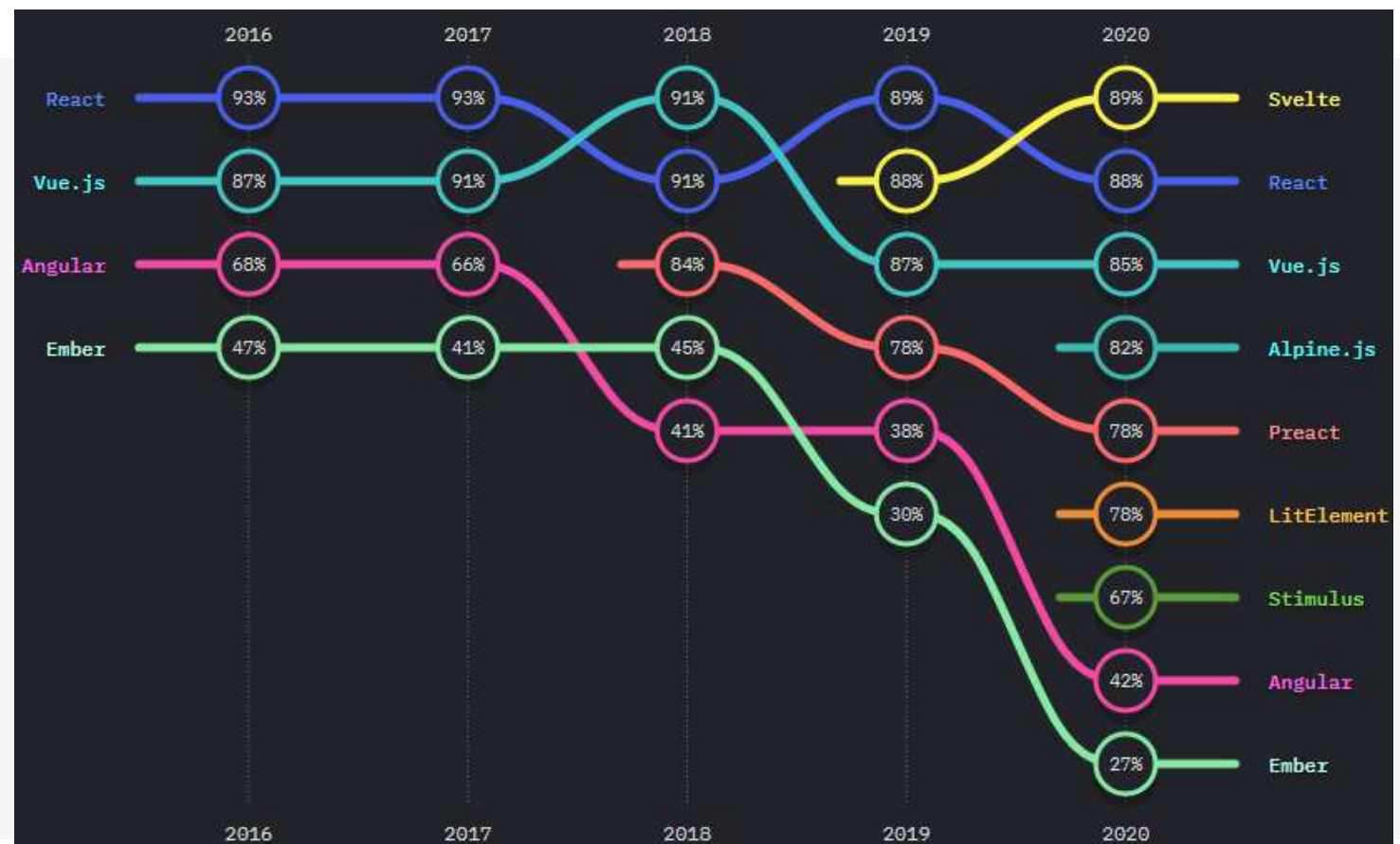
Classement des frameworks  
Front-End les plus populaires  
suivant les ratios **d'intérêt**



# Frameworks javaScript : satisfaction

JavaScript Developer Survey - Edition 2020 (<https://2020.stateofjs.com/fr-FR/technologies/front-end-frameworks/>)

Classement des frameworks Front-End les plus populaires suivant les ratios de **satisfaction**



# Le langage javaScript

- **ECMAScript :**

- Ensemble de normes standardisées par l'organisation européenne Ecma International pour l'utilisation des langages de programmation du type Script.





# Rappel : JavaScript ES2015 (ES6)

- **Déclaration et portée des variables :**
  - **let, const** : définir des variables et des constantes
- **Structures conditionnelles :**
  - **if ... else if ... else**
  - **variable = condition ? expressionIf : expressionElse;**
- **Structures itératives :**
  - **for .. in** : avec les objets - `for (let key in object)`
  - **for .. of** : avec les tableaux - `for (let v of array)`
- **Fonctions :**
  - **Fonctions nommées** : `function sum(n1, n2)`
  - **Fonctions anonymes** : `let sum = function (n1, n2)`
  - **Fonctions fléchées** : `let f = (name) => "hello" + name;`

# Rappel : JavaScript ES2015 (ES6)

- **Méthodes de manipulation des tableaux :**

- Remplir un tableau : **fill**
- Inverser et trier un tableau : **reverse** et **sort**
- Ajouter des éléments en queue du tableau : **concat**
- Extraire un sous-tableau : **slice**
- Copier et remplacer au sein d'un tableau : **copyWithin**
- Trouver le premier et le dernier indice d'une valeur : **indexOf** et **lastIndexOf**
- Ajouter/supprimer un élément :
  - en queue du tableau : **push** et **pop**
  - en tête du tableau : **unshift** et **shift**
  - à partir de n'importe quelle position : **splice**

```
let t = [0, 1, 2, 3, 4];
t.fill(0); // 0, 0, 0, 0, 0
t.reverse(); // 4, 3, 2, 1, 0
t.concat(4, 30); // 4, 3, 2, 1, 0, 4, 30
t.slice(2, 4); // 2, 1
t.indexOf(3); // 1
t.push(5); // 4, 3, 2, 1, 0, 4, 30, 5
t.pop(); // 4, 3, 2, 1, 0, 4, 30
t.shift(); // 3, 2, 1, 0, 4, 30
t.unshift(-1); // -1, 3, 2, 1, 0, 4, 30
t.splice(3,1);
-- supprime 1 élément à partir de l'indice 3
// -1, 3, 2, 0, 4, 30
const months= ['Jan','March','April','June'];
months.splice(1, 0, 'Feb');
-- ajoute à l'indice 1 Feb
// ["Jan", "Feb", "March", "April", "June"]
months.splice(4, 1, 'May');
-- remplace 1 élément à l'indice 4 par May
// ["Jan", "Feb", "March", "April", "May"]
```

# Rappel : JavaScript ES2015 (ES6)

- **Map :**

- Parcourir les éléments d'un tableau.

- **Reduce :**

- Combiner les valeurs : réduire les valeurs en une seule valeur

- **Filter :**

- Filtrer les éléments d'un tableau suivant une certaine condition.

```
let jeux = ['go', 'de'];  
let prix = [99, 15];  
let t = jeux.map((j, i) => ({ nom: j, prix: prix[i] }));  
// [{ nom: 'go', prix: 99 }, { nom: 'de', prix: 15 }]
```

```
let t = [0, 1, 2, 3];  
let somme = t.reduce((a, b) => a + b);  
console.log("Somme=" + somme);
```

```
let t3 = t.filter((valeur) => {  
    return valeur < 8;  
});
```

```
let pets = ['cat', 'dog', 'fish'];  
let nb = pets.map(s => s.length).reduce((a, b) => a + b);  
console.log('nb = ' + nb);
```



# Déstructuration d'affectation

- Pour les tableaux, on peut copier les valeurs dans des variables individuelles nommées en une seule instruction.

```
let t = [1, 2, 3];  
let [x, y] = t;  
x; // 1  
y; // 2  
z; // error  
let t = [1, 2, 3, 4, 5];  
let [x, y, ...rest] = t;  
x; // 1  
y; // 2  
rest; // [3, 4, 5]  
let x = 5, y = 10;  
[x, y] = [y, x];  
x; // 10  
y; // 5
```

# Déstructuration d'objets

- Pour les objets, il faut respecter le nom des champs (peu importe l'ordre).

```
let o = { b: 2, c: 3, d: 4 };  
let { a, b, c } = o;  
a; // undefined  
b; // 2  
c; // 3  
d; // erreur
```

- Lorsque la déstructuration n'est pas effectuée au moment de la déclaration, il faut ajouter des parenthèses.

```
let o = { b: 2, c: 3, d: 4 };  
let d, b, c;  
{ d, b, c } = o; // erreur  
({ d, b, c } = o); // ok  
console.log(d, b, c); // 4 2 3
```

# Introduction à JSX

- **JSX** : Extension de syntaxe pour JavaScript qui ajoute la syntaxe XML à JavaScript.
- On peut utiliser React sans JSX, mais JSX rend React beaucoup plus élégant.
- JSX est décomposé comme suit : **JS** pour JavaScript & **X** pour XML.
- Tout comme XML, les balises JSX ont un nom de balise, des attributs et des enfants.
- Parmi les points communs avec XML, on citera notamment :
  - Sensible à la casse : `<monJsx />` n'a pas le même sens que `<MonJSX />`.
  - Il est obligatoire de fermer une balise toute seule : par exemple `<input />`  
**Mais pas** `<input>`.
  - Il faut respecter l'ordre d'ouverture et de fermeture d'une balise (on ferme dans l'ordre inverse de l'ouverture) : `<p><span>JSX</span></p>` **Mais pas**  
`<p><span>JSX</p></span>`.

# JSX

- **Déclaration des variables et constantes**

```
const PI = <h1>3.14</h1>;  
var name = 'Tout le monde';
```

- **Utiliser des expressions dans JSX**

```
const name = 'Tout le monde';  
const element = <h1 count={1 + 2 + 3 + 4} >Bonjour, {name}</h1>;
```

- **Les propriétés** : On ne parle plus d'attributs HTML mais de propriétés d'un composant : Chaque balise HTML est en réalité un composant à part entière qui accepte des propriétés dites : **props**.

```
<input  
  type="number" // String  
  max={100} // La valeur 100  
  onChange={this.handleChange} // Fonction  
>
```

# HTML VS JSX

---

HTML	JSX
for	htmlFor
class	className
<style color ="blue">	< style={{color:'blue'}}>
<!--Comment-->	{ /*Comment*/ }



# HTML VS JSX

- **Convertir en ligne HTML en JSX :**
  - <https://magic.reactjs.net/htmltojsx.htm>

## HTML to JSX Compiler

☒ Create class · Class name: NewComponent

### Live HTML Editor

```
<!-- Hello world -->
<div class="awesome" style="border: 1px solid red">
  <label for="name">Enter your name: </label>
  <input type="text" id="name" />
</div>
<p>Enter your HTML here</p>
```

```
var NewComponent = React.createClass({
  render: function() {
    return (
      <div>
        { /* Hello world */ }
        <div className="awesome" style={{border: '1px solid red'}}>
          <label htmlFor="name">Enter your name: </label>
          <input type="text" id="name" />
        </div>
        <p>Enter your HTML here</p>
      </div>
    );
  }
});
```

# JS VS JSX

- **JSX** : Souplesse dans l'écriture, la lecture et la compréhension du code.

## Sans JSX 😞

```
React.createElement(  
  'div',  
  null,  
  React.createElement('h2', null, 'Title List'),  
  React.createElement('p', null, 'Contenu')  
)
```

## Avec JSX 😊

```
const element = () => (  
  <div>  
    <h2>Title List</h2>  
    <p>Contenu</p>  
  </div>  
)
```

# JS VS JSX

- **Convertir en ligne JSX en JS :**
  - <https://infoheap.com/online-react-jsx-to-javascript/>

## Input jsx

example:

```
ReactDOM.render(<h1>Hello world</h1>, document.getElementById('hello'));
```

```
ReactDOM.render(  
  <h1>Hello world</h1>,  
  document.getElementById('hello'));
```

convert above jsx to javascript

(char count: 77)

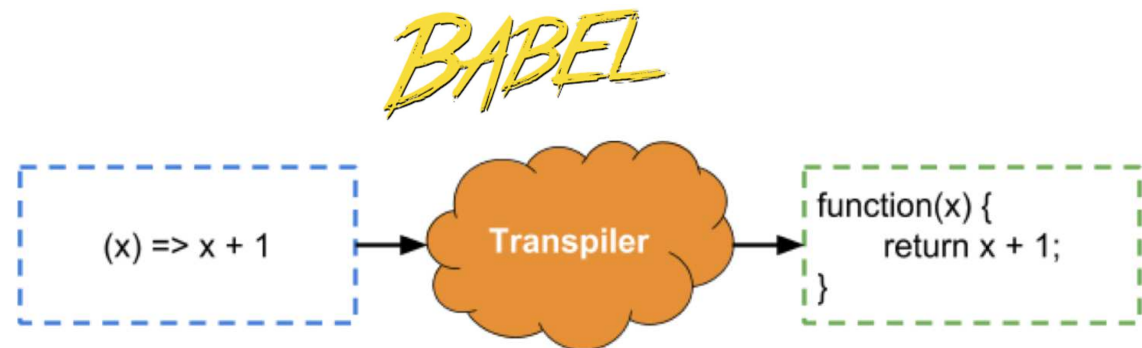
jsx is valid (input char count: 72)

## Converted javascript

```
ReactDOM.render(React.createElement(  
  'h1',  
  null,  
  'Hello world'  
) , document.getElementById('hello'));
```

# Babel

- **Babel** : transpileur ECMAScript 6. Il permet de compiler le **JSX** en code **JavaScript** classique et exécutable par le navigateur.
- Il permet d'utiliser les nouvelles fonctionnalités du langage sans devoir attendre leur support par les navigateurs.
- Babel sait compiler et traduire les éléments suivants :
  - les raccourcis de fonction ;
  - les fonctions asynchrones ;
  - les classes ;
  - la déstructuration ;
  - let ;
  - etc.



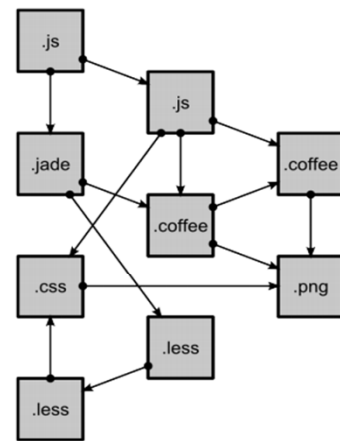
# Webpack

- **Webpack :**

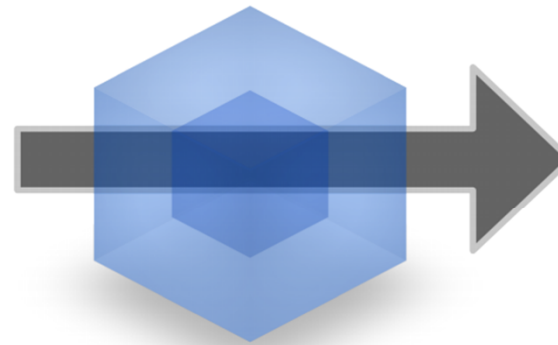
- Il dispose d'un système de "loaders" qui vont permettre d'inclure de nouveaux types de fichiers ou d'appliquer des transformations spécifiques pour que le code soit compréhensible directement par le navigateur.

Découper le code sous forme de modules qui seront ensuite fusionnés en un seul fichier

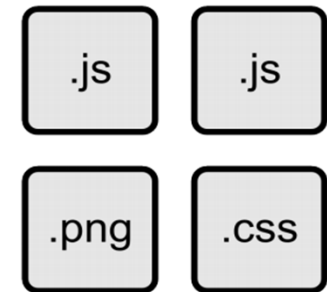
→ Outil incontournable dès lors que l'on travaille sur des projets JavaScript complexes



modules  
with dependencies



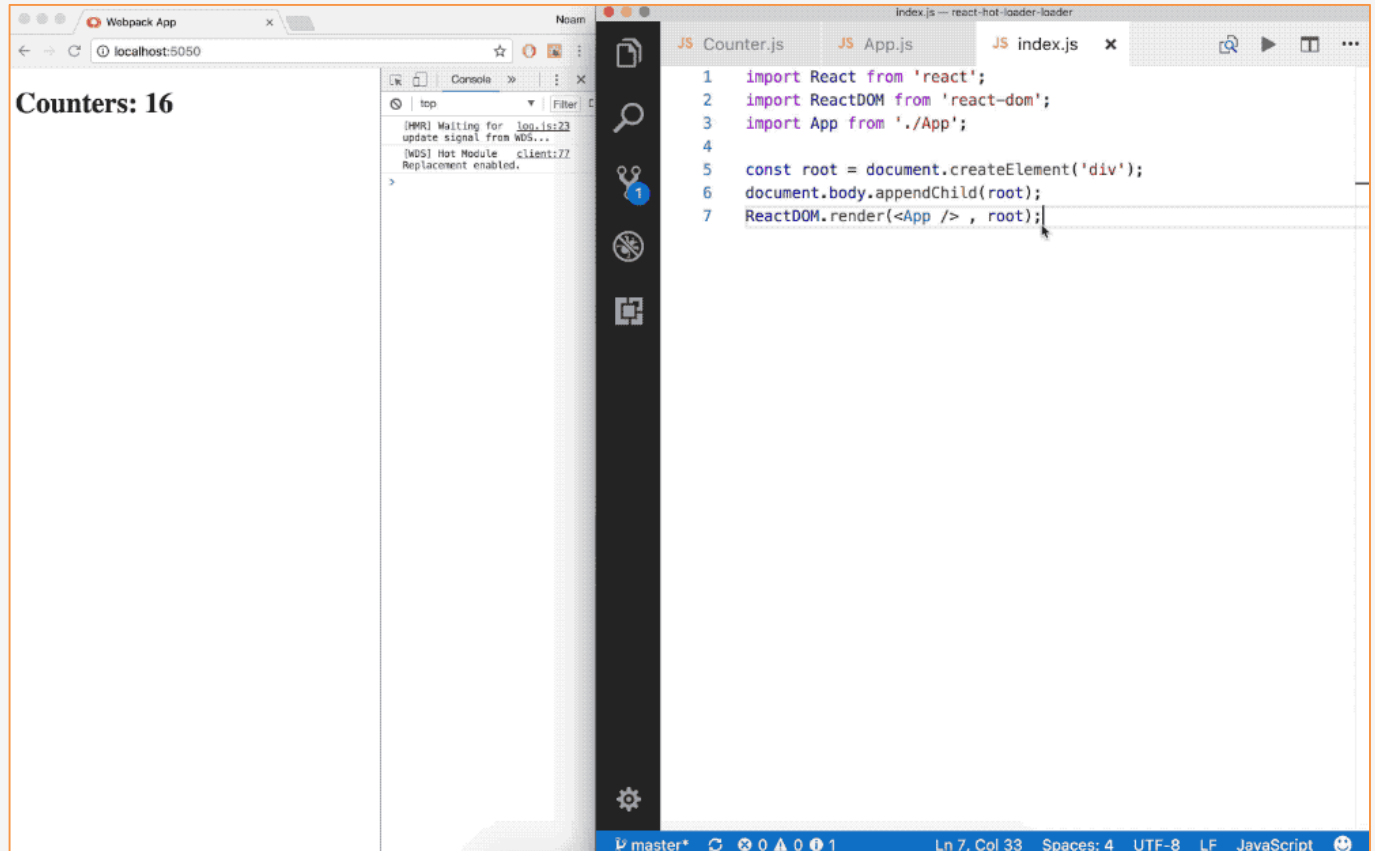
webpack  
MODULE BUNDLER



static  
assets

# Rechargement à chaud (Hot Loader)

- Le Rechargement à chaud améliore l'expérience de développement en mettant à jour automatiquement les modules dans le navigateur lors de l'exécution sans nécessiter une actualisation ou recharge complète de la page.
- **Exemple** : React Hot Loader permet de faire fonctionner React avec le « *hot module replacement* » de webpack.



# React : présentation générale

- **React :**

- **Bibliothèque** (framework) JavaScript, développée et pilotée par **Facebook** depuis 2013,
- Projet open-source, désormais distribué sous la licence MIT,
- React se concentre principalement sur la gestion de **l'interface utilisateur (UI)** :
  - Pas de notions de contrôleurs, services, directives ou modèles. On ne parle que de la couche Vue
  - Les autres couches applicatives (routage côté client, le stockage des données, etc.) sont laissées aux solutions complémentaires de son écosystème
  - Exemples : React-Router, Reduc, Redux-offline, ... .
- Sites web développés avec React et React Native : Atlassian, Dailymotion, Dropbox, Instagram, Netflix, Paypal, Twitter, Wordpress, Yahoo, ...

# React : présentation générale

---

- **Quels langages utilise React ?**

- **HTML** pour les vues
- **CSS** pour les styles
- **JSX** (JavaScript XML) : pour les scripts

- **Quelques outils utilisés par React :**

- **npm** (node package manager) : le gestionnaire de paquets par défaut pour une application JavaScript
- **webpack** : bundler JavaScript (pour construire le graphe de dépendances et regrouper des ressources de même nature (.js ou .css...) dans un ou plusieurs bundles)

- **Documentation officielle :**

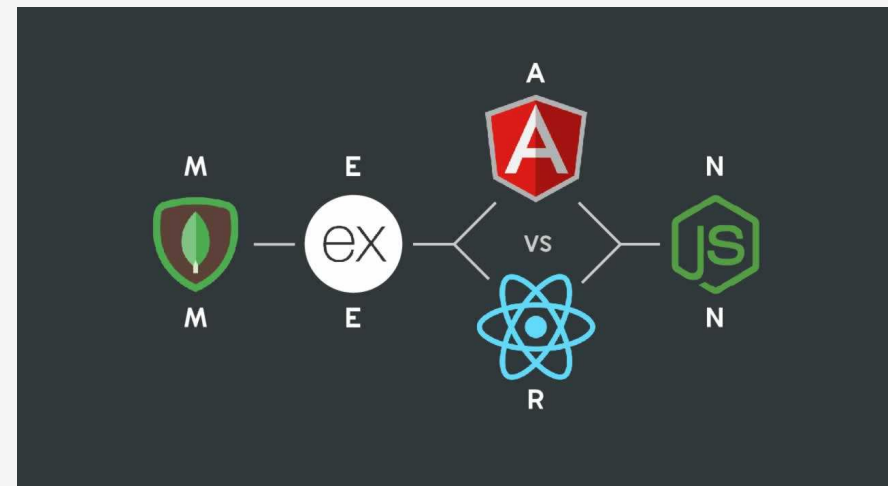
- <https://fr.reactjs.org/>



# Développement web : MEAN & MERN Stack

- **MEAN & MERN Stack** : alliance de technologies (frameworks) Javascript open source utilisée pour :
  - rendre le processus de développement fluide et facile.
  - donner la capacité aux [développeurs full stack](#) de développer un site de A à Z sans avoir à faire intervenir une autre compétence : développeur à tout faire

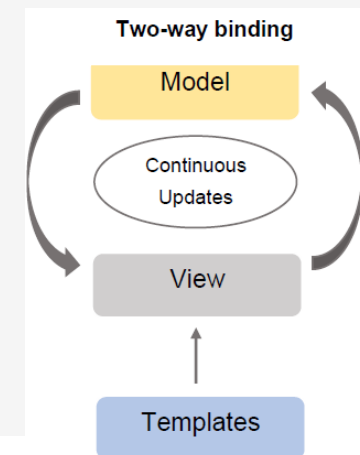
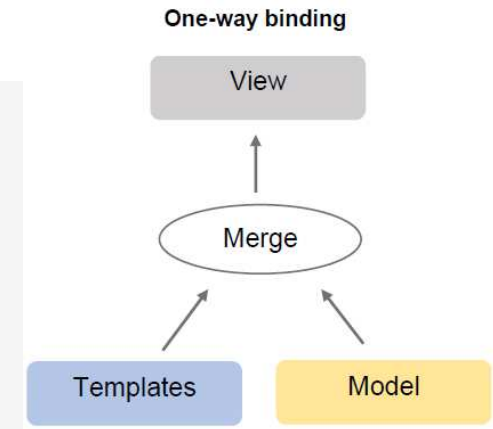
- **M** : Mongo DB
- **E** : Express
  - **A** : Angular
  - **R** : React
- **N** : Node JS



# Développement web : MEAN & MERN Stack

## • Angular Vs React :

	Angular	React
Type	Framework	Librairie
Auteur	google	Facebook
Apparition	Octobre 2010	Mai 2013
Langage	TypeScript	JavaScript (JSX)
Data binding	Bi-directionnel	Uni-directionnel
Du web au mobile	Ionic	React Native
Usage	Sony, nike, Genral motors, HBO, ...	Netflix, whatsapp, dropbox, instagram, ...



# Développement web : MEAN & MERN Stack

---

- **Choisissez React si :**

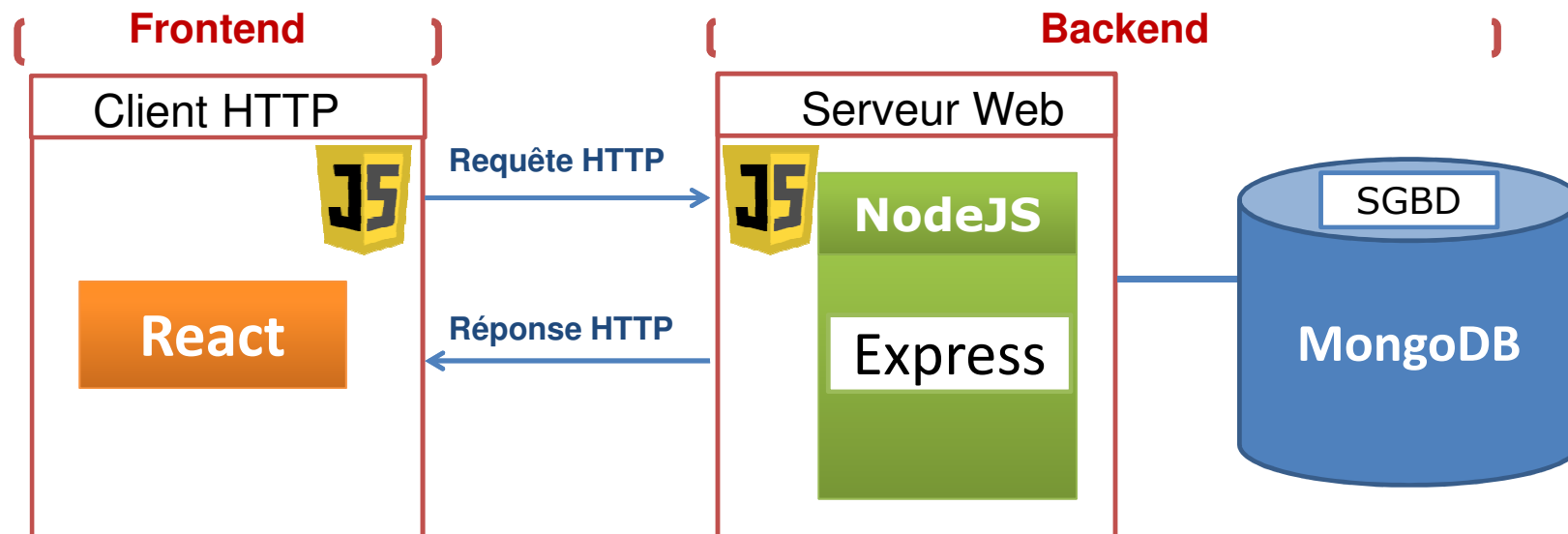
- Vous recherchez une solution hautement personnalisée dans le projet.
- Votre projet est une application d'une seule page, comme une application de discussion en ligne, une application de visualisation de données.
- Vous envisagez de créer ensuite une application mobile multiplate-forme à l'aide de React Native.

- **Choisissez Angular si :**

- Vous construisez une application Web à page unique à grande échelle.
- L'application va être riche en fonctionnalités et contenir du contenu dynamique.
- Votre objectif est un projet à long terme.
- Vous envisagez de créer une application Web hybride ou progressive au lieu d'une application mobile native ou multiplateforme.

# React pour un développement MERN Stack

- Architecture MERN Stack :



# Environnement du travail et Installation

- **Installation des outils :**

- **Node JS :**

- [Site : https://nodejs.org/en/download/](https://nodejs.org/en/download/)
    - Node JS installe l'outil npm (Node Package Manager) qui permet de télécharger et installer des bibliothèques Java Script : gestionnaire de modules de Node.
    - Pour vérifier la version installée :

- > `node --version` ou `node -v`

- **Create React App (CRA) :** `create-react-app`

- Outil pour faciliter le développement d'applications web fondées sur React
    - Eviter les problèmes d'installation, de configuration et d'intégration
    - Génération automatique d'un squelette applicatif

# Environnement du travail et Installation

- **Installation des outils :**

- **Editeur :**

- Visual Studio Code (<https://code.visualstudio.com/>)
    - ou encore IntelliJ, WebStorm, PHP Storm, ...

- **Extensions :**

- Installation des plugins (extensions) y associés est souhaitable afin de faciliter le codage

Exemple : Simple React Snippets sous VSC

- imr : import React
      - imrc : Import React / Component
      - cc : Class Component
      - ccc : Class Component with Constructor
      - rfc : React fonctionnal Component
      - rfce : React Functional Component with Export
      - ...

# Lancement d'un projet React

- **Etapas à suivre :**

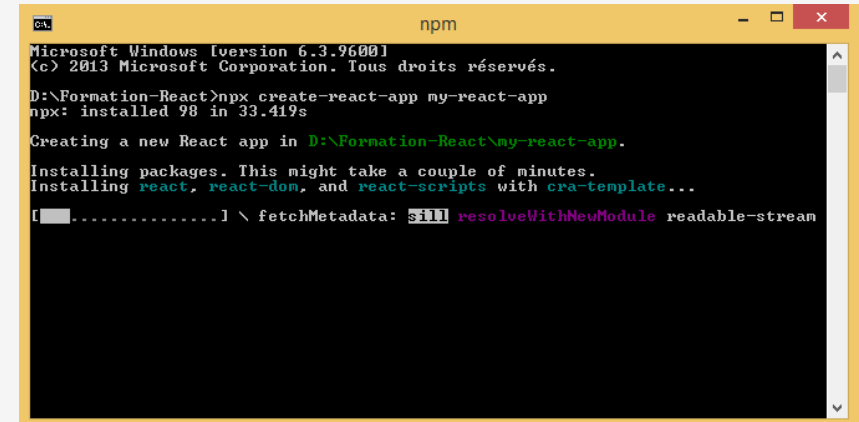
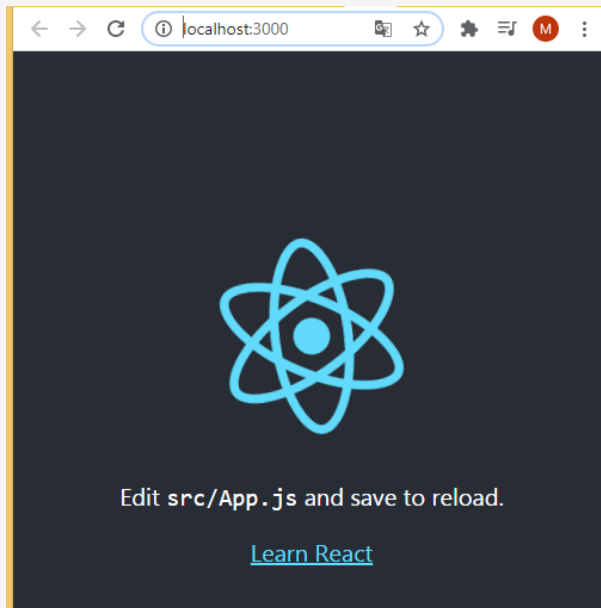
- Créer un nouveau dossier « D:/Formation-React »
  - > **npx create-react-app my-react-app**

> **cd my-react-app**

> **npm start**

L'application se lance par défaut en local sur :

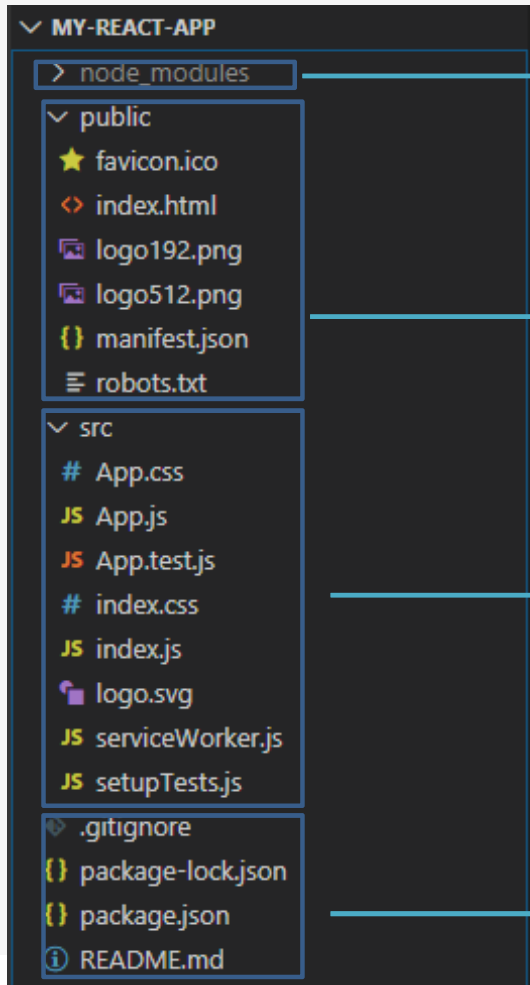
<http://localhost:3000/>

A screenshot of a terminal window titled "npm". It shows the command "npx create-react-app my-react-app" being executed. The output indicates that the command was successful, creating a new React app in the directory "D:\Formation-React\my-react-app". It also shows the installation of packages (react, react-dom, and react-scripts) using the cra-template.

Au préalable, pour installer l'outil :  
**create-react-app :**

> **npm install -g create-react-app**

# Structure d'une application ReactJS



Dépendances externes (librairies) de l'application

Ressources publiques de l'application

Code source de l'application (composants du react : jsx, css, ...)

Fichiers de configuration du projet

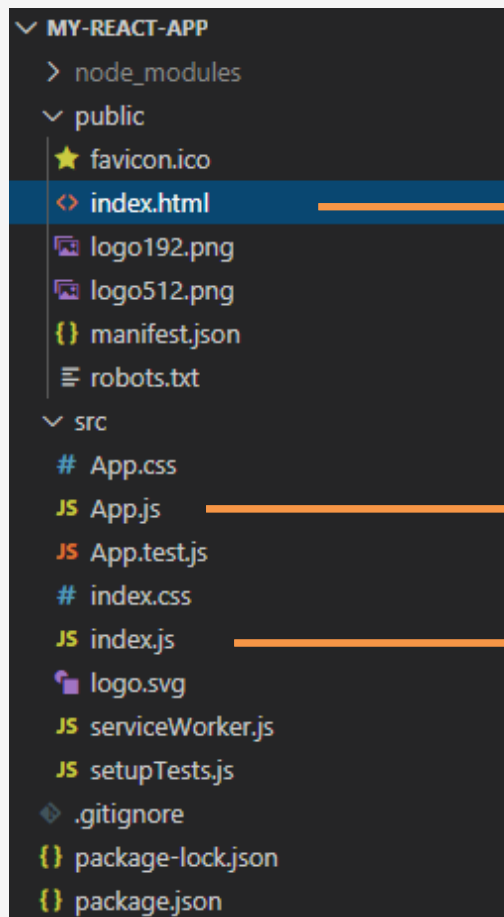


# Structure d'une application ReactJS

- **Principaux répertoires et fichiers d'une application React :**

- **Nodes\_modules** : Toutes les bibliothèques Java Script externes utilisées par l'application
- **index.html** : le point d'entrée de l'application
- **App.js** : le composant principal (root component)
- Au niveau de CRA (Create React Application)
  - Seuls les deux fichiers **src/index.js** et **public/index.html** sont exigés
  - Le reste des fichiers est optionnel
- **Package.json** : contient les métadonnées du projet (titre, version, ...) ainsi que toutes les dépendances nécessaires afin de pouvoir créer et exécuter une application React
- Le répertoire **public/** : fichiers HTML, JSON et images de base (les racines de l'application).
- Le répertoire **src/** : fichiers sources (js, css, etc.) que nous créerons devront être dans
- De préférence, tous les composants (components) qui vont être définis par la suite devront être dans un dossier à part (à titre d'exemple : **components**).

# Structure d'une application ReactJS



(1)

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

(2)

(3)

```
function App() {
  return (
    <h2>Bienvenue à cette formation React</h2>
  );
}
export default App;
```

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

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

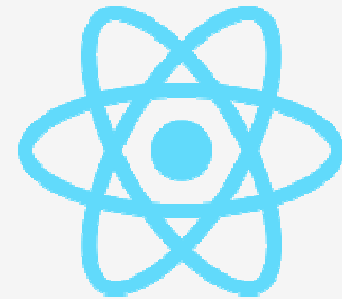
- **Atelier 1 :**
  - ❑ Mise en place de l'environnement de développement & Initiation à JSX.

# Développer avec React JS

# Développer avec React JS : Principes de base

---

- ❑ Les fondamentaux du React
- ❑ Architecture React
  - DOM
  - Virtual DOM
- ❑ Principe de data binding
- ❑ Les composants React
  - ❑ Notion du state
  - ❑ Composants statefull et stateless



 **Atelier :** Définition et création de simples composants React

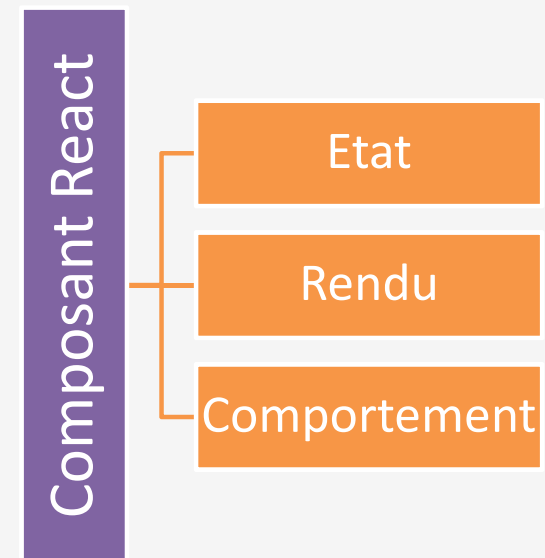
# Les fondamentaux du React

- **React :**

- permet de créer des applications web de type SPA (Single Page Application)
- basé sur une approche de programmation orientée composants (Web Components)

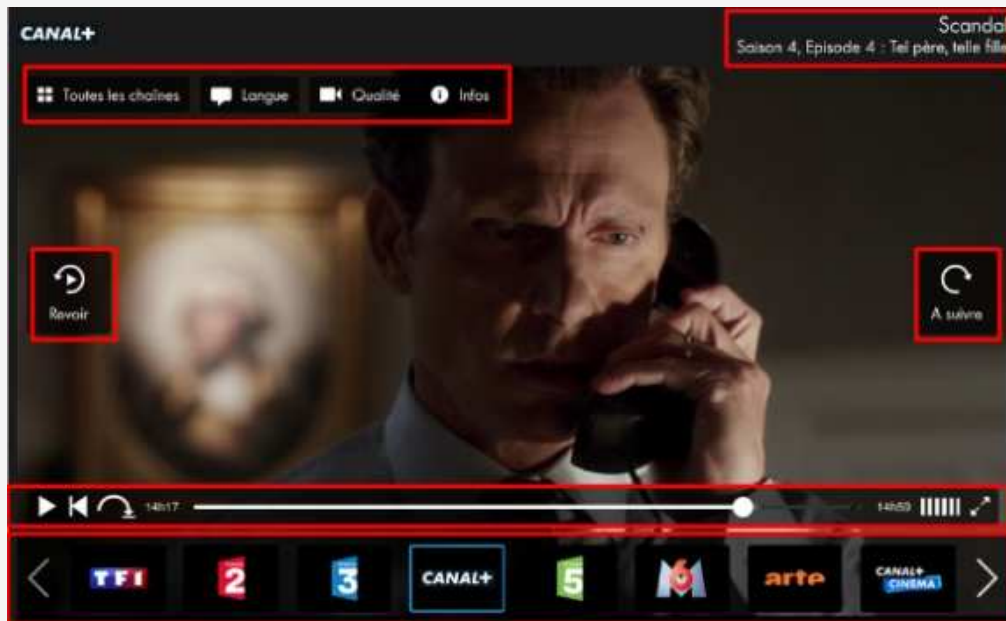
- **Composant React est caractérisé par :**

- L'**état** du composant (State) qui représente le Modèle
- Le **rendu** : Structure de la vue du composant (View)
- Le **comportement** (Behaviour ou Controller)





# Présentation générale

- **React** : Arbre de composants
  - Le tout n'est que composant !
    - Et un composant est composable.



## React : avantages et limites

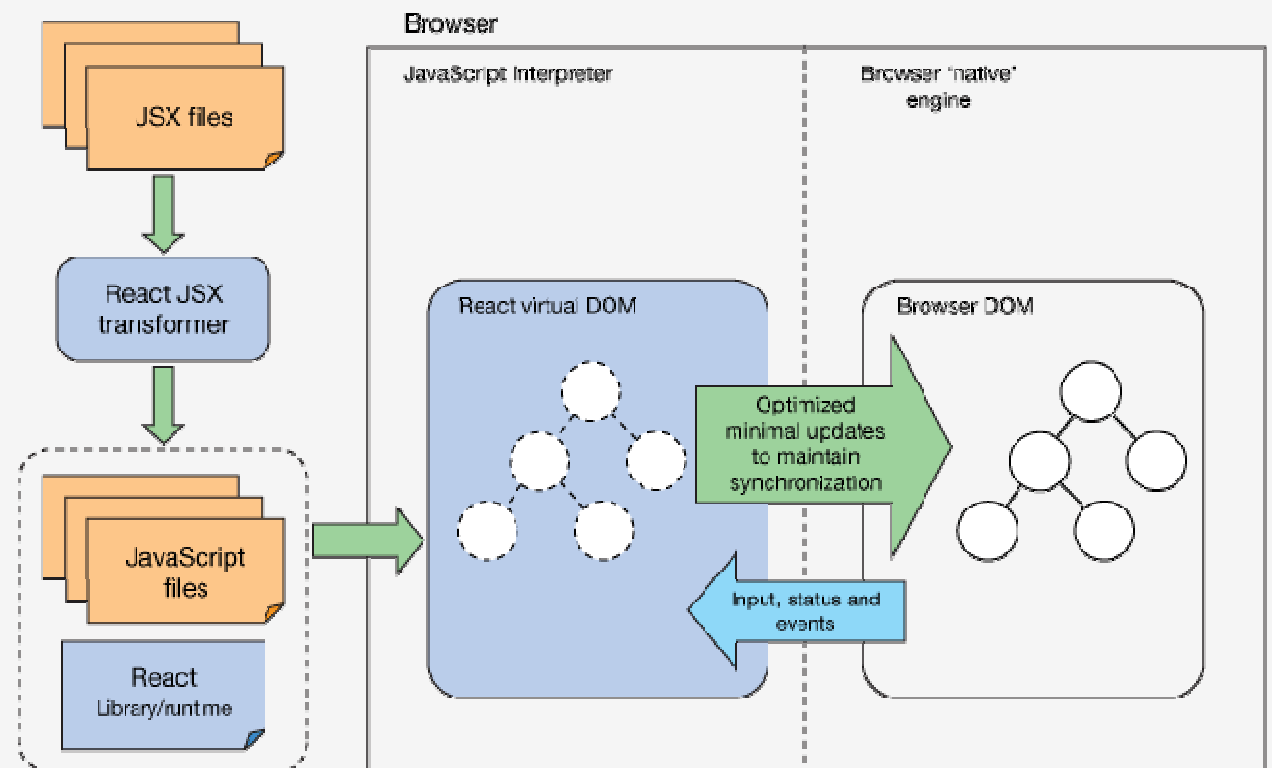
---

 <b>Avantages</b>	 <b>Limites</b>
Il augmente les performances de l'application.	React est juste une bibliothèque, pas un framework complet
Il peut être facilement utilisé côté client ainsi que côté serveur.	Sa bibliothèque est très grande et prend du temps à comprendre
Grâce à JSX, la lisibilité du code augmente	Il peut être difficile pour les programmeurs débutants de le comprendre
React est facile à intégrer avec d'autres frameworks comme Meteor, Angular, etc.	Le codage devient complexe car il utilise les modèles en ligne et JSX



# Architecture React

- Le moteur de rendu JSX, permet de coupler dans un même fichier JSX, les trois aspects d'un composant React
- Le moteur de React génère un DOM virtuel qui sera transformé et synchronisé avec le DOM réel du Navigateur
- Le Virtuel DOM optimise les changements à apporter au niveau du Browser DOM. Ce qui lui permet d'accélérer l'aspect réactif du rendu.



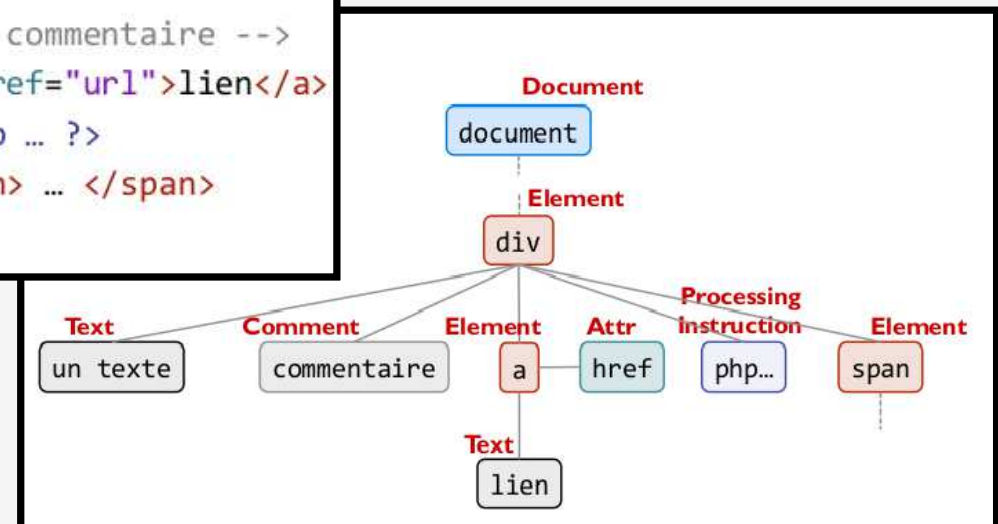
# Rappel - DOM : Document Object Model

- Le DOM est une interface de programmation qui permet à des scripts d'examiner et de modifier le contenu du navigateur web.
- Un document XML ou HTML peut être vu sous la forme d'un arbre.
- Chaque nœud de l'arbre est considéré comme un objet.

• Les principaux types de nœuds sont:

- **Document** : la racine
- **Element** : les balises standards
- **Text** : le texte
- **Comment** : les commentaires
- **Attr** : les attributs des balises
- **Processing instructions** : les balises spécifiques comme par exemple `<?php...?>`

```
<div>un texte  
  <!-- commentaire -->  
  <a href="url">lien</a>  
  <?php ... ?>  
  <span> ... </span>  
</div>
```



# DOM : Document Object Model

---

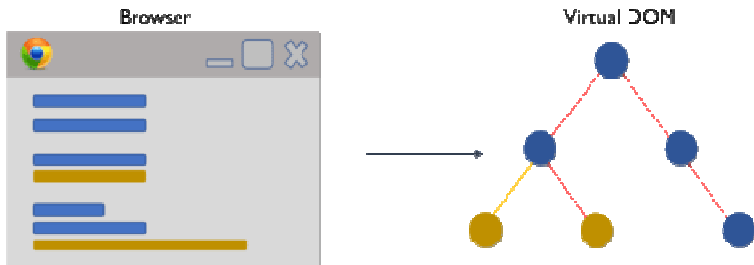
- Le DOM HTML est une norme standard qui définit :
  - Les éléments HTML comme des objets
  - Les propriétés de tous les éléments (les valeurs qu'on peut définir)
  - Les méthodes pour accéder aux éléments
  - Les événements qu'on peut effectuer sur les éléments,
- Le DOM offre des fonctions appropriées afin de trouver les éléments :

Méthode	Description
<code>document.getElementById(id)</code>	Trouve un élément par son identificateur
<code>document.getElementsByTagName(nom)</code>	Trouve des balises via leur nom
<code>document.getElementsByClassName(nom)</code>	Trouve des balises via le nom de leur class CSS

# Virtual DOM

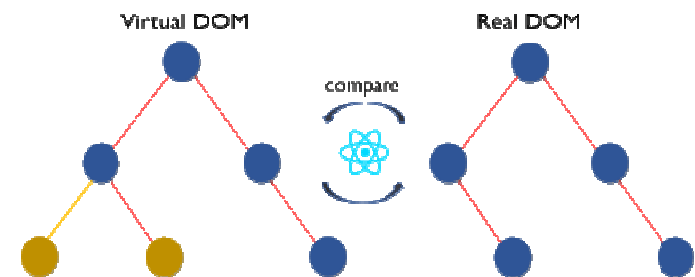
## Etape 1 :

- Chaque fois que des données changent, l'interface utilisateur entière est restituée en représentation Virtual DOM



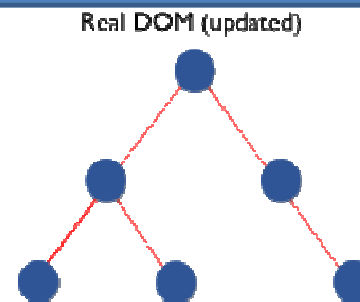
## Etape 2 :

- La différence entre la représentation DOM précédente et la nouvelle est calculée



## Etape 3 :

- Une fois les calculs effectués, le DOM réel va mettre à jour uniquement les objets qui ont réellement changé.



## Real DOM Vs Virtual DOM

---

Real DOM	Virtual DOM
Il se met à jour lentement.	Il se met à jour plus rapidement.
Peut directement mettre à jour le code HTML	Impossible de mettre à jour directement le code HTML.
Crée un nouveau DOM si l'élément est mis à jour.	JSX s'occupe de la mise à jour de l'élément
La manipulation DOM est très coûteuse.	La manipulation DOM est très facile.
Gaspillage de la mémoire.	Bonne gestion de la mémoire.

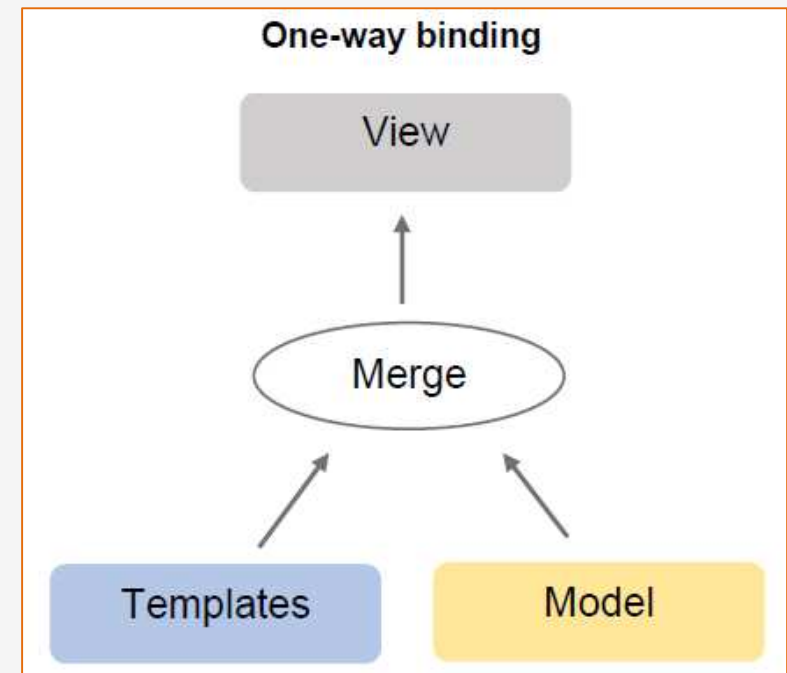
# Principe de data binding

---

- **Data binding** : (liaison de données)
  - Ce mécanisme consiste à lier la partie **vue** à la partie **logique** : si la source de données change, il est possible de faire en sorte que le contrôle soit automatiquement mis à jour.
  - Grâce au data binding les éléments du code HTML seront liés au contrôleur JavaScript
  - Ceci passe généralement par les étapes suivantes :
    - Détection des changements
    - La résolution des liaisons associées
    - La mise à jour du DOM

# Principe de data binding

- **Data binding** : (liaison de données)
  - **React** : une liaison de données à sens unique (**one-way**)
    - Tout d'abord, l'état du modèle est mis à jour, puis il rend la modification de l'élément d'interface utilisateur.
    - Toutefois, si vous modifiez l'élément d'interface utilisateur, l'état du modèle ne change pas.



# Les composants React

- **Composants React :**

- Ils servent le même objectif que les fonctions JavaScript, mais fonctionnent de manière isolée et renvoient du HTML via une fonction de rendu.
- **render()** : fonction de base de React pour afficher du contenu à l'écran,
- Ils peuvent être définis sous forme d'une fonction ou d'une classe
- Ils sont indépendants et réutilisables.

## Composant purement fonctionnel (stateless)

```
function App() {  
  return (  
    <h2>Bienvenue à cette formation React</h2>  
  );  
}  
export default App;
```

## Composant de classe (statefull) généralement contenant un état

```
class App extends React.Component {  
  render(){  
    return (  
      <h2>Bienvenue à cette formation React</h2>  
    );  
  }  
}  
export default App;
```



# Les composants React : Notion du state

- C'est quoi le « state » en React ?
  - objet, où l'on va pouvoir stocker les données relatives à un composant donné
  - accessible uniquement à travers son composant
  - va être utilisable uniquement dans les composants "stateful"
- Création du state :
  - **this.state** = { définition *des données sous forme de (key : value)* }

```
// On va créer notre state dans la méthode constructor() de notre composant
constructor(props) {
  super(props); // Nous permet d'utiliser les méthode de React.Component
  this.state = {
    // On stocke ici notre data
    monTexte: "Bienvenue à cette formation React."
  };
}
```

# Les composants React : Notion du state

- Utilisation du « state » :

- L'accès aux données stockées dans le state se fait à travers : ***this.state.nomAttribut***
- Pour l'affichage, on utilise la notation { } (on parle d'interpolation)

```
class App extends Component {  
  // On va créer notre state dans la méthode constructor() de notre composant  
  constructor(props) {  
    super(props); // Nous permet d'utiliser les méthodes de React.Component  
    this.state = {  
      // On stocke ici notre data  
      monTexte: "Bienvenue à cette formation React."  
    };  
  }  
  render() {  
    return (<h2>{this.state.monTexte}</h2>);  
  }  
}  
export default App;
```

# Les composants React : Notion du state

---

- **Modification du « state » :**

- Se fait via la méthode ***this.setState*** à laquelle on passera un objet avec l'attribut à modifier et la nouvelle valeur à appliquer.

```
this.state={  
  monTexte: ''  
}
```

```
this.setState(  
  {  
    monTexte:"Bonjour à toutes et à tous !"  
  });
```

# Les composants React : Notion du state

- **Modification du « state » :**

- Définition de l'action au click :
  - Pour cet exemple : on met à jour l'état en incrémentant la valeur courante du compteur
  - React effectue un rendu intégral d'un composant à chaque changement des données qui lui sont associées,

```
handleClick = () => {  
  this.setState({  
    value : this.state.value + 1  
  })  
}
```

```
<input type="button" onClick={()=>this.handleClick()} value="Incrémenter"/>  
  
<h2>{this.state.value}</h2>
```

# Les composants React : Notion du map

- **Notion de map :**

- méthode afin de parcourir d'une manière simple les éléments d'un tableau (Array)
- Transformer un tableau d'objets en un tableau d'éléments JSX

- **Exemple 1 :**

```
const nombres = [1, 2, 3];  
const double = nombres.map((number) => number * 2);
```

- **Exemple 2 :**

```
this.state = {  
  // On stocke ici notre data  
  monTexte: "Bienvenue à cette formation React",  
  personnes:[  
    {id:1, nom:"John"},  
    {id:2, nom:"Blandine"},  
    {id:3, nom:"Paul"}  
  ]  
};
```

```
render() {  
  return (<div>  
    <h2>{this.state.monTexte}</h2>  
    {  
      this.state.personnes.map((p)=>  
        <p> ID : {p.id} --- NOM : {p.nom}</p>)  
    }  
  </div>)  
}
```

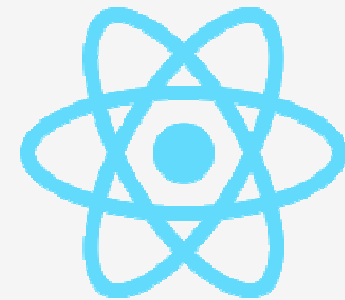
- **Atelier 2 :**
  - Définition et création de simples composants React

# **Communication inter-composants React**

# Développer avec React JS : Interactivité des composants

---

- ☐ Notion de props
- ☐ Cycle de vie
- ☐ Introduction aux hooks
- ☐ Gestion des événements & formulaires
- ☐ Composition et réutilisation des composants
- ☐ Rendu conditionnel
- ☐ Routage
- ☐ Récupération des données à partir d'une API Rest



 **Atelier :** Création d'un ensemble structuré de composants React



# Les composants React : Notion de props

## • Notions de props :

- Abréviation de « properties » pour contenir les informations relatives à un composant.
- Technique qui consiste à partager du code entre des composants React
- La plupart des composants peuvent être personnalisés avec différents **props** lors de leur création.
- A la base, **props** est un objet. Il stocke les valeurs des **attributs** (Attributes) d'une étiquette (Tag) :
  - arguments transmis aux composants React via des attributs Html
  - arguments **transmis par le parent** et accessibles via : **this.props**

```
function Project() {  
  return (<div>  
    {this.props.task}  
  </div>);  
} export default Project;
```

```
const element = <Project task="Conception"/>;  
ReactDOM.render(  
  element,  
  document.getElementById('root')  
>);
```

# Les composants React : Notion de props

- Exemple :

*index.js*

```
import React from 'react';
import ReactDOM from 'react-dom';

class Personne extends React.Component {
  render() {
    return (<div>
      <h2>Prénom : {this.props.prenom} </h2>
      <h2>Nom : {this.props.nom} </h2>
    </div>)
  }
}

const myElement = <Personne prenom = "Anis" nom = "ASSAS" />;
ReactDOM.render(myElement, document.getElementById('root'));
```

# Les composants React : Notion de props

- Le même exemple : une autre façon de faire

*personne.js*

```
import React from 'react';
class Personne extends React.Component {
  render() {
    return (<div>
      <h2>Prénom : {this.props.prenom} </h2>
      <h2>Nom : {this.props.nom} </h2>
    </div>) }
} export default Personne;
```

*App.js*

```
import Personne from './personne';
class App extends React.Component {
  render() {
    const myElement=<Personne prenom = "Anis" nom = "ASSAS"/>;
    return (
      <div> {myElement} </div>
    ); }
}
```

# Les composants React : Notion de props

- Remarques importantes :



- props** : propriétés immuables : on ne peut pas affecter des valeurs ou des objets à 'props' (chose qu'on peut faire par contre au niveau de 'state').

- On peut affecter des valeurs par défaut aux **props** via : **defaultProps**

*personne.js*

```
import React from 'react';
class Personne extends React.Component {
  render() {
    return (<div>
      <h2>Prénom : {this.props.prenom} </h2>
      <h2>Nom : {this.props.nom} </h2>
    </div>) }
}
Personne.defaultProps = { prenom : "Foulen",
                           nom : "Foulenii" }
export default Personne;
```

```
<Personne prenom = "Anis" nom = "ASSAS"/>
<Personne prenom = "Ahmed"/>
<Personne nom = "Ben Ahmed"/>
<Personne />
```

# Cycle de vie d'un composant React

- Cycle de vie d'un composant :

Quand une instance de composant est créée et insérée dans le DOM

## Montage

- `constructor()`
- `componentWillMount()`
- `render()`
- `componentDidMount()`



Quand un composant est mis à jour ou "re-rendered" : son state ou props ou son composant parent est modifié

## Mise à jour

- `componentWillReceiveProps()`
- `componentWillUpdate()`
- `render()`
- `componentDidUpdate()`



Quand un composant est retiré du DOM

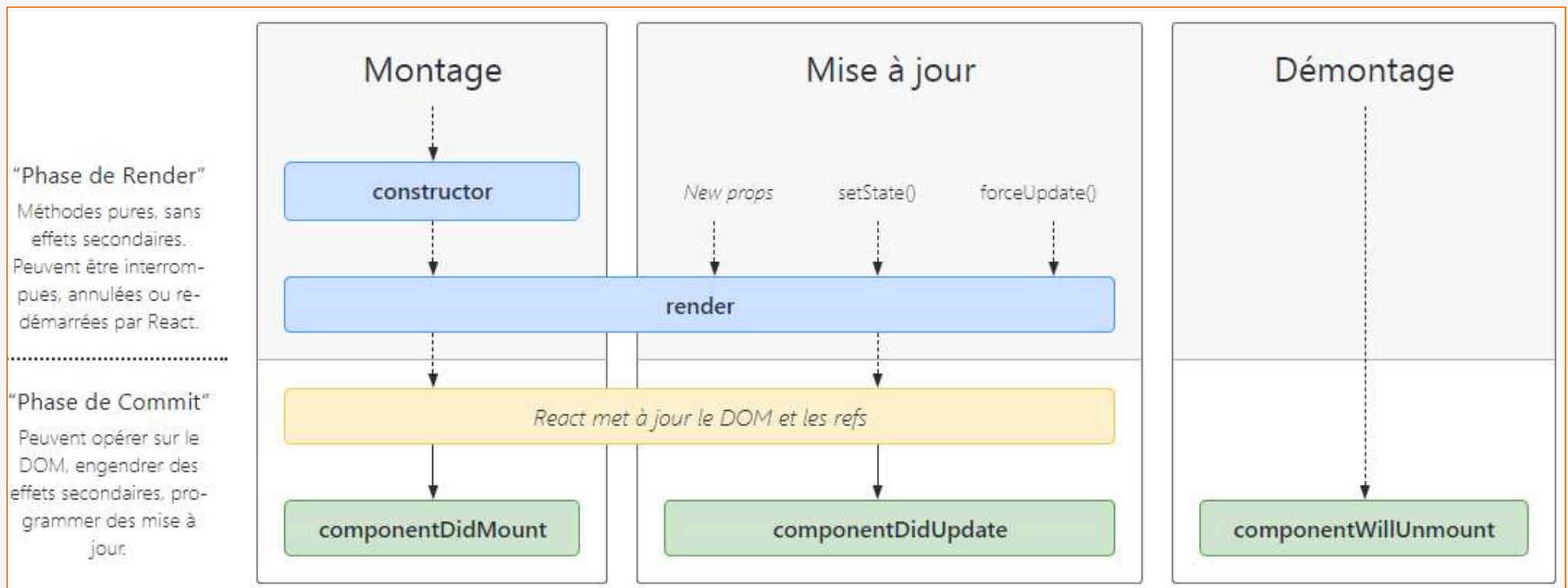
## Démontage

- `componentWillUnmount()`

# Cycle de vie d'un composant React

- **Cycle de vie d'un composant :**

- <https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/>



# Cycle de vie d'un composant React

- **Description des différentes méthodes :**

- **Montage (mounting) :**

- **componentWillMount()** : le code sera exécuté juste avant que le composant soit monté
    - **componentDidMount()** : le code sera exécuté dès que le composant aura été monté

- **Mise à jour (updating) :**

- **componentWillUpdate()** : le code sera exécuté juste avant que le composant soit mis à jour
    - **componentDidUpdate()** : le code sera exécuté après que le composant ait été mis à jour
    - **componentWillReceiveProps()** : le code sera exécuté juste avant que la nouvelle valeur d'une *prop* soit passée au composant

- **Démontage (unmounting) :**

- **componentWillUnmount()** : le code sera exécuté juste avant que le composant soit démonté et retiré du DOM

# Cycle de vie d'un composant React

---

- **Remarques complémentaires :**

- Y a d'autres méthodes de cycle de vie (qui restent peu utilisées) :
  - `shouldComponentUpdate()`, `getDerivedStateFromProps()`, `getSnapshotBeforeUpdate()`, `getDerivedStateFromError()`, ...
- Certaines méthodes sont considérées dépréciées. Pour éviter les avertissements dans les nouveaux codes, on pourrait ajouter le préfixe `UNSAFE_` :
  - `UNSAFE_componentWillMount()`
  - `UNSAFE_componentWillUpdate()`
  - `UNSAFE_componentWillReceiveProps()`



# Cycle de vie d'un composant React

- Exemple : Montage (mounting)

```
class Color extends React.Component {
  constructor(props) {
    super(props);
    this.state = { favoriteColor : "red" }
    console.log(this.state.favoriteColor);
  }
  UNSAFE_componentWillMount() {
    console.log(this.state.favoriteColor);
    setTimeout(()=> {
      this.setState({favoriteColor: this.props.favCol});
    }, 5000);
  }
  componentDidMount() {
    setTimeout(()=> {
      this.setState({favoriteColor: "blue"})
    }, 10000);
  }
  render() {
    console.log(this.state.favoriteColor);
    return(<h1>My Favourite color is : {this.state.favoriteColor}</h1>);
  }
}export default Color;
```

```
class App extends React.Component {
  render() {
    return (
      <div>
        <Color favCol="yellow"/>
      </div>
    );
  }
} export default App;
```

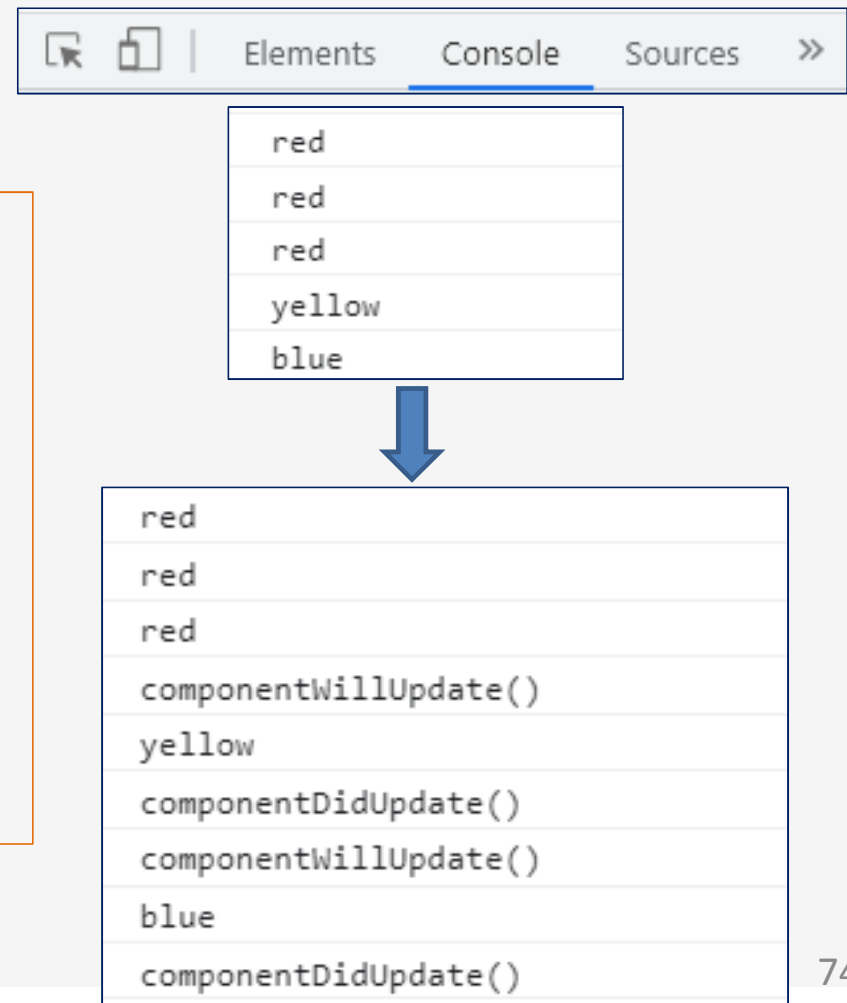


**My Favorite Color is ?????**

# Cycle de vie d'un composant React

- Exemple : Mise à jour (updating)

```
class Color extends React.Component {  
  . . .  
  UNSAFE_componentWillUpdate() {  
    console.log("componentWillUpdate()");  
  }  
  componentDidUpdate() {  
    console.log("componentDidUpdate()");  
  }  
  . . .  
}
```



# Cycle de vie d'un composant React

- **Exemple :**

- **Démontage (unmounting)**

```
class Color extends React.Component {
  componentWillMount() {
    console.log("componentWillUnmount() - "+this.state.favoriteColor)
  }
  render() {
    console.log(this.state.favoriteColor);
    return ( <div>
      <h1>My Favourite color is : {this.state.favoriteColor} </h1>
      </div>);
  }
} export default Color;
```

```
render() {
  return (
    <div>
      <Color favCol="green"/>
    </div>
  );
}
```

red
red
red
componentWillUpdate()
blue
componentDidUpdate()
componentWillUpdate()
blue
componentDidUpdate()
red
red
red
componentWillUnmount() -blue
componentWillUpdate()
green
componentDidUpdate()
componentWillUpdate()
blue
componentDidUpdate()

# Introduction aux Hooks

---

- **Qu'est ce qu'un Hook ?**

- **Fonction** qui permet de bénéficier d'un état local et d'autres fonctionnalités de React sans le recours d'une classe.
- Les *Hooks* sont arrivés avec React 16.8.

- **Pourquoi les Hooks ?**

- Hook : Functional component
  - Pas de *class*
  - Pas de *constructor*
  - Pas de *this*
  - Beaucoup plus simple à lire (on évite le mode verbeux des classes)
  - Manipulation beaucoup plus simple et performante des événements et changements de props.

# Le Hook d'état

- **Le Hook d'état :**

- **useState** : hook qui permet d'ajouter l'état local React à des fonctions composants.

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

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```
import React, { useState } from 'react';  
  
function Example() {  
  const [count] = useState(0);  
}
```

Déclaration d'une nouvelle variable  
d'état **count** initialisée à 0



# Utilisation du Hook d'état

---

- Lire l'état

```
// valeur actuelle du compteur dans une classe  
<p>Compteur : {this.state.count} </p>
```

```
// valeur actuelle du compteur via un hook  
<p>Compteur : {count} </p>
```

- Mettre à jour l'état

```
// mise à jour de la valeur dans une classe  
this.setState({ count: this.state.count + 1 })
```

```
// mise à jour de la valeur dans la fonction  
setCount(count + 1)
```

# Utilisation du Hook d'effet

- Le Hook d'effet `useEffect()` :

permet de définir une action à effectuer dès que le composant est affiché ou mis à jour (tout comme les méthodes de cycle de vie des classes React).

```
import React, {useState, useEffect} from 'react'

function Example() {
  const [count, setCount] = useState(0);
  // Similaire à componentDidMount et componentDidUpdate :
  useEffect(() => {
    // Met à jour le titre du document via l'API du navigateur
    document.title = `Vous avez cliqué ${count} fois`;
  });
  return (<div>
    <p>Vous avez cliqué {count} fois</p>
    <button onClick={() => setCount(count + 1)}> Cliquez ici </button>
  </div> );
}
```

# Gestion des événements

- **Événements React :**

- Tout comme HTML, React peut effectuer des actions en fonction des événements utilisateur.
- React a les mêmes événements que HTML: onClick, onChange, onMouseOver, etc.

## ❏ Exemple 1 :

```
<button onClick={this.handleClick}>Cliquer</button>

handleClick(){
  alert("Bonjour à toutes et à tous...")
}
```



**NB :** En React, l'appel de la fonction se fait plutôt comme suit : **{this.handleClick}**

Et non pas : ~~{this.handleClick()}~~



# Gestion des événements

- **Binding :**

- Pour les méthodes dans React, le mot clé **this** doit représenter le composant propriétaire de la méthode
- On devrait préciser à une fonction quelle valeur utiliser pour la variable **this**.

```
<button onClick={this.handleClick}>Cliquer</button>  
handleClick(){  
  alert(this)  
}
```



localhost:3000 indique  
undefined

OK

# Gestion des événements

- **Solutions :**

- 1<sup>ère</sup> solution : utiliser **bind(this)**

```
<button onClick={this.handleClick.bind(this)}>Cliquer</button>
```

- 2<sup>ème</sup> solution : utiliser une fonction fléchée pour garder la même valeur de this

```
<button onClick={() => this.handleClick()}>Cliquer</button>
```

- 3<sup>ème</sup> solution : déclarer la fonction comme une fonction fléchée :

```
handleClick = () => {  
  alert(this)  
}
```

localhost:3000 indique

[object Object]

OK

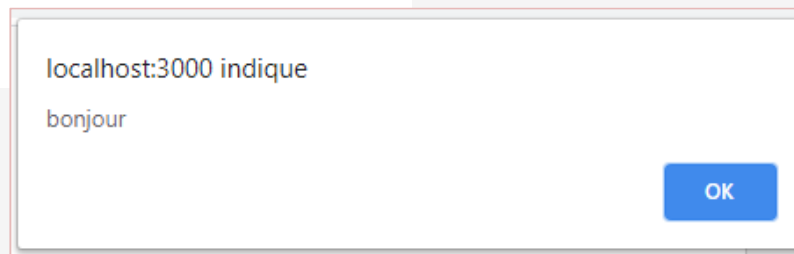
# Gestion des événements

- **Passage des arguments :**

- A part **this**, il y a aussi la possibilité de passer d'autres arguments au niveau de l'appel de la fonction comme suit : **bind(this, argument)**

```
<button onClick={this.handleClick.bind(this, "bonjour")}>Cliquer</button>
```

```
handleClick (msg) {  
  alert(msg)  
}
```



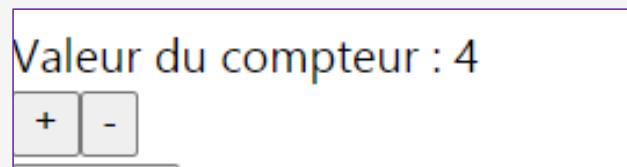
**Remarque :**

- Si on envoie les arguments sans utiliser la méthode bind, la fonction sera exécutée lorsque la page sera chargée sans attendre que le bouton soit cliqué.

# Gestion des événements

- **Exercice d'application :**

- Gestion d'un compteur
  - Déclarer dans state la propriété 'compteur' initialisé à 0
  - Concevoir la page afin de créer deux boutons + et – afin de pouvoir incrémenter et décrémenter le compteur
  - A chaque action, afficher par conséquent la valeur courante du compteur



Valeur du compteur : 4

+

-

# Gestion des événements

- Corrigé : Extraits du code source

```
<div>
  Valeur du compteur : {this.state.compteur}
</div>
<div >
  <button onClick={()=>this.compute('+')} >+</button>
  <button onClick={()=>this.compute('-')} >-</button>
</div>
```

```
this.state = {
  compteur: 0,
};
```

```
compute=(op)=> {
  let sign=op==='+'? 1:-1;
  let c=this.state.compteur+sign;
  this.setState({
    compteur:c,
  });
}
```

# Gestion des formulaires

---

- **Composants de formulaires en React :**

- Tout comme en HTML, React utilise des formulaires pour permettre aux utilisateurs d'interagir avec la page Web.
- Au niveau des formulaires en React, on utilise simplement la *prop* **value=**, quel que soit le type du composant.
- Quand il s'agit de valeurs multiples, comme par exemple pour **<select multiple>** et ses enfants **<option>**, on pourrait faire recours à un tableau de valeurs à props.

# Gestion des formulaires

- **Composants de formulaires en React :**

- React utilise le concept de « composants contrôlés » pour les formulaires.
  - Accès à la valeur du champ : ***event.target.value***

```
this.state = {  
  . . .  
  personne: ''  
};  
  
<form>  
  <h2>Bonjour {this.state.personne}</h2>  
  <input type="text" onChange={this.handleChange}/>  
  <button>Ajouter personne</button>  
</form>
```

```
handleChange = (event) => {  
  this.setState({personne : event.target.value});  
}
```

**Bonjour Anis**

Anis

# Gestion des formulaires

- **Soumission d'un formulaire : onSubmit**

- Contrôler l'action de soumission en ajoutant un gestionnaire d'événements dans l'attribut **onSubmit**.

```
<form onSubmit={this.addPerson}>
  <h2>Bonjour {this.state.personne}</h2>
  <input type="text" onChange={this.handleChange}/>
  <button>Ajouter personne</button>
</form>
```

```
addPerson = (event) => {
  event.preventDefault();
  alert("" + this.state.personne);
}
```

localhost:3000 indique  
Anis

OK

**Bonjour Anis**

Anis

Ajouter personne



**event.preventDefault** : Empêcher le rechargement / rafraîchissement du navigateur



# Gestion des formulaires

- Ajout d'un élément à une liste :

- Contrôler l'action de soumission en ajoutant un gestionnaire d'événements dans l'attribut *onSubmit*.

```
this.state = {  
  monTexte: "Bienvenue à cette formation React",  
  personnes: ["Ali", "Salah", "Mohamed"],  
  personne: ''  
};
```

```
addPerson= (event) => {  
  event.preventDefault();  
  let tab_personnes= this.state.personnes.slice();  
  tab_personnes.push(this.state.personne);  
  this.setState({  
    personnes:tab_personnes  
  })  
}
```

```
this.setState({  
  personnes:[...this.state.personnes,this.state.personne]  
})
```

( Autre façon de faire )



**slice** : Créer une copie du tableau

# Gestion des formulaires

- **TextArea :**
  - Les input *TextArea* utilisent plutôt un attribut value

```
render()
{
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Description :
        <textarea value={this.state.value} onChange={this.handleChange}/>
      </label>
      <input type="submit" value="Submit"/>
    </form>
  );
}
```

# Gestion des formulaires

- **Selected :**

- Pour être conforme avec les autres types de formulaires, React utilise l'attribut **value** au lieu **selected** pour déterminer l'élément sélectionné.

```
render() {  
  return (  
    <form onSubmit={this.handleSubmit}>  
      <label>  
        Choisissez votre framework préféré:  
        <select value={this.state.value} onChange={this.handleChange}>  
          <option value= "Angular">Angular</option>  
          <option value= "React">React</option>  
          <option value= "Vue">Vue.js</option>  
        </select>  
      </label>  
      <input type="submit" value="Submit" />  
    </form>  
  );  
}
```

# Composition et réutilisation des composants

- **Composition :**

- Les composants peuvent faire référence à d'autres composants.

```
function App() {  
  return (<div>  
    <Welcome name="Mohamed" />  
    <Welcome name="Salah" />  
    <Welcome name="Ali" />  
  </div>)  
}
```

```
function Welcome(props) {  
  return (  
    <div>  
      <h1>Hello, {props.name}</h1>  
    </div>  
  );  
}  
export default Welcome;
```

# Rendu conditionnel

- React traite les conditions de la même façon que JavaScript.
- Utiliser des opérateurs JavaScript comme l'opérateur **if** pour créer des éléments représentant l'état actuel et laisser React mettre à jour l'interface utilisateur pour les faire correspondre.

```
function Bonjour(props) {  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <Connected />;  
  }  
  return <Guest />;  
}
```

```
ReactDOM.render(  
  <Bonjour isLoggedIn={false} />,  
  document.getElementById('root')  
)
```

```
function Connected(props) {  
  return <h1>Bienvenue à notre site !</h1>;  
}
```

```
function Guest(props) {  
  return <h1>Veuillez vous inscrire !</h1>;  
}
```

# Routage

- Le mécanisme qui permet de naviguer d'une page à une autre sur un site web :
  - router clairement le "flux de données" (data flow) de l'application
  - permet de définir des URL dynamiques et de sélectionner un Component approprié pour afficher son rendu sur le navigateur d'utilisateur en correspondance à chaque URL.
- React ne propose pas de module traitant du routing. On fait appel à une librairie externe.

- **Installation :**

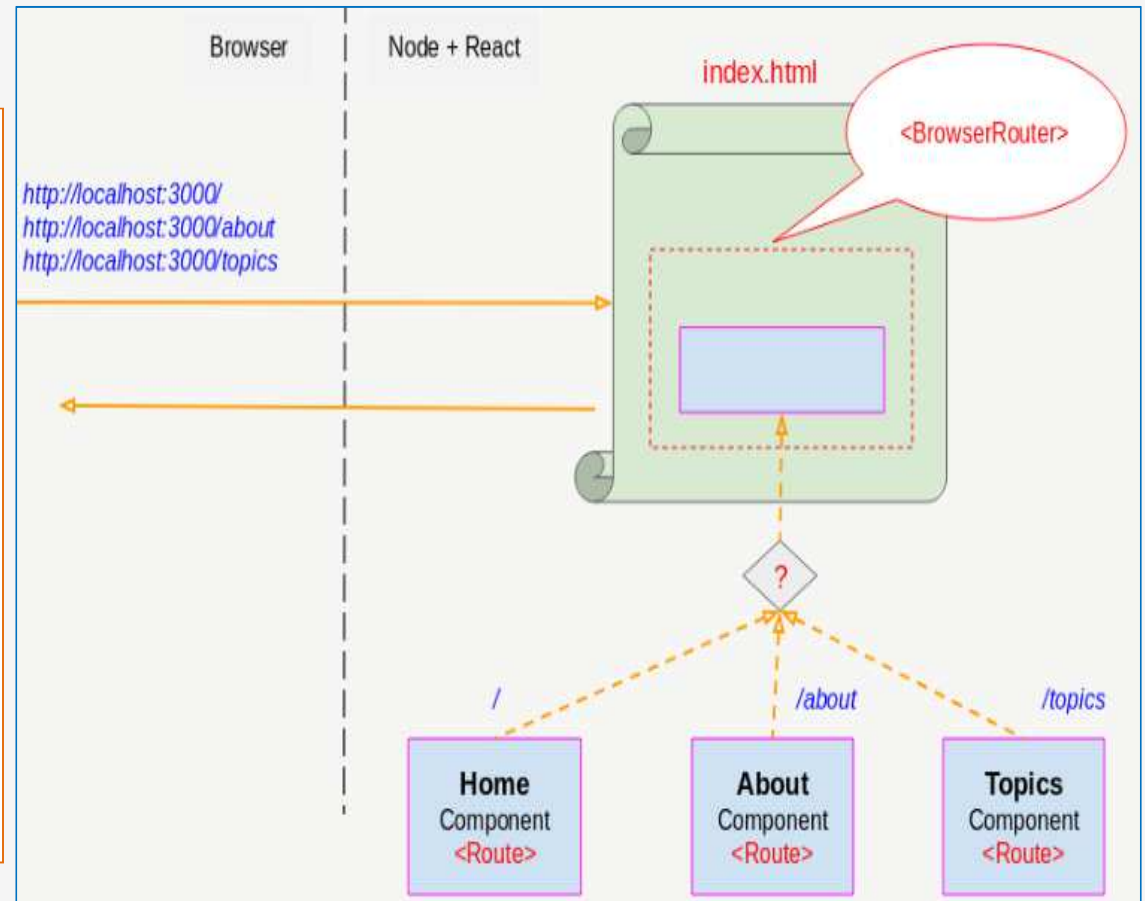
```
> npm install --save react-router-dom
```

```
import {  
  BrowserRouter as Router,  
  Route,  
  Link,  
  Switch,  
} from 'react-router-dom'
```

# Routage

## • Utilisation :

```
render(){  
  return (  
    <Router>  
      <nav>  
        <ul>  
          <li><Link to="/">Home</Link> </li>  
          <li><Link to="/about">About</Link></li>  
          <li><Link to="/topics">Topics</Link></li>  
        </ul>  
      </nav>  
      <div>  
        <Switch>  
          <Route exact path="/" component={Home}></Route>  
          <Route path="/about" component={About}></Route>  
          <Route path="/topics" component={Topics}></Route>  
        </Switch>  
      </div>  
    </Router>  
  )  
}
```



# Récupération des données à partir d'une API

- **axios :**

- Bibliothèque JavaScript fonctionnant comme un client HTTP permettant de communiquer avec des API en utilisant des requêtes
- basée sur des promesses pour le navigateur et node.js
- Intercepter et transformer les données de demande et de réponse
- Transformations automatiques pour les données JSON

- **axios.get(url) :** retourner une promesse de la réponse de la requête auprès de l'url donné.

- **Installation :**

> `npm install axios`



# Récupération des données à partir d'une API

- **Exemple :**

- Api de test : <https://jsonplaceholder.typicode.com/users>

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

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

- Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdottir V
- Glenna Reichert
- Clementina DuBuque

- **Atelier 3 :**

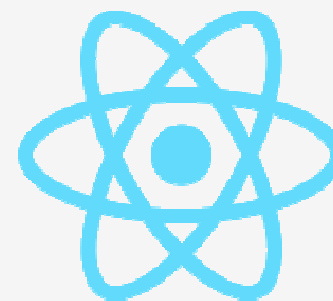
- ☐ Création d'un ensemble structuré de composants React.

# Redux

# Redux : Plan

---

- ❑ Introduction
- ❑ Flux
- ❑ Redux
  - ❑ C'est quoi ?
  - ❑ Principe
  - ❑ Concepts clés
- ❑ Connexion d'un composant React à Redux



⚙️ **Atelier** : Développement d'une application SPA avec ReactJS et Redux.

# Design Patterns

---

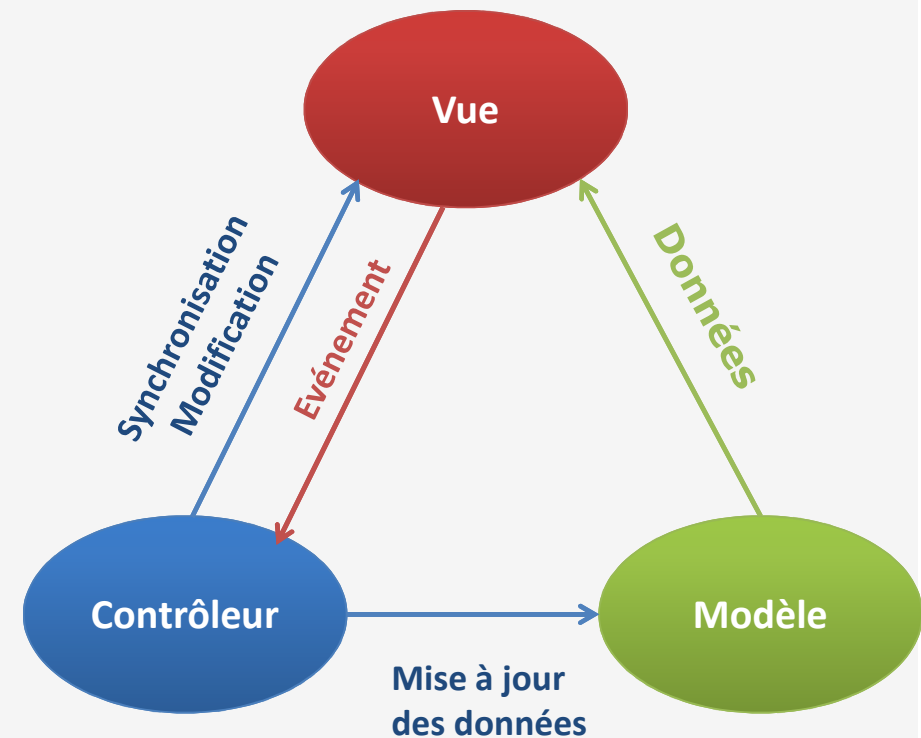
- **Patrons ou Modèles de conception** : recueil de bonnes pratiques de conception pour un certain nombre de problèmes récurrents en programmation orientée objet.
- Ces modèles de conception sont définis pour pouvoir être utilisés avec un maximum de langages orientés objets. Il est habituel de regrouper ces modèles communs dans trois grandes catégories :
  - ***les modèles de création*** (creational patterns)
  - ***les modèles de structuration*** (structural patterns)
  - ***les modèles de comportement*** (behavioral patterns)
- Le motif de conception le plus connu est sûrement **le modèle MVC** (Model View Controller) mis en œuvre en premier avec SmallTalk.

# Architecture MVC

Architecture de développement très répandue visant à séparer le code source distinctement en 3 couches :

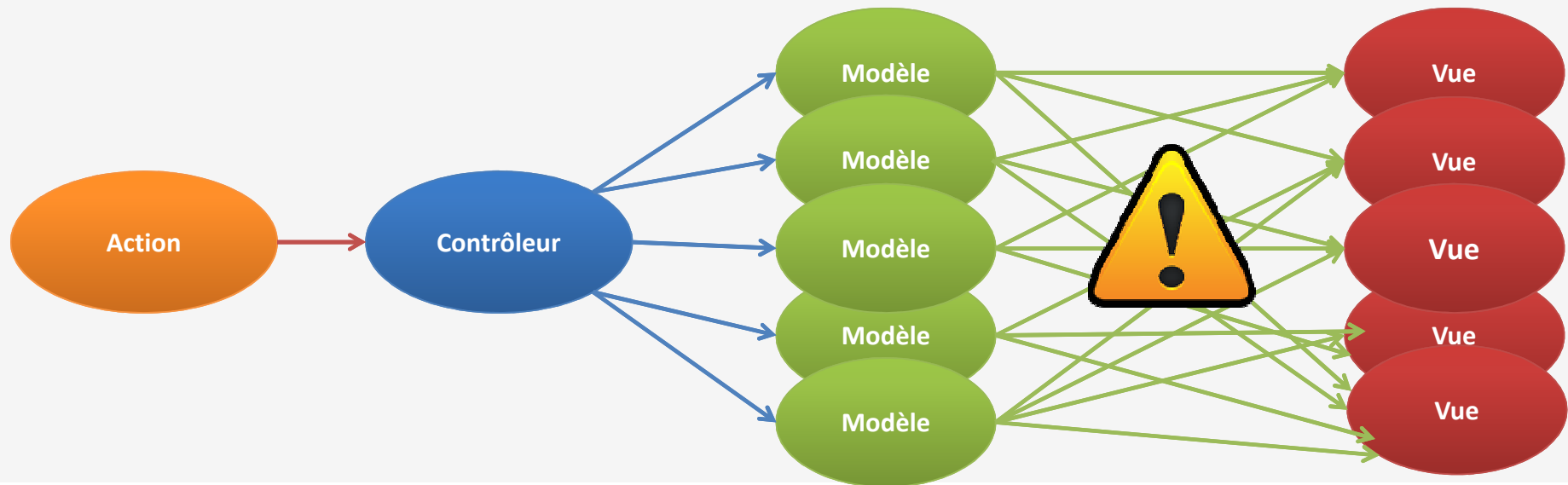
- **Le modèle** : les données utilisées par l'application.
- **La vue** : l'interface utilisateur, la façon dont les informations seront affichées à l'écran.
- **Le contrôleur** : chargé de la synchronisation du modèle et de la vue : reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.

Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle et ensuite avertit la vue que les données ont changé pour que celle-ci se mette à jour.



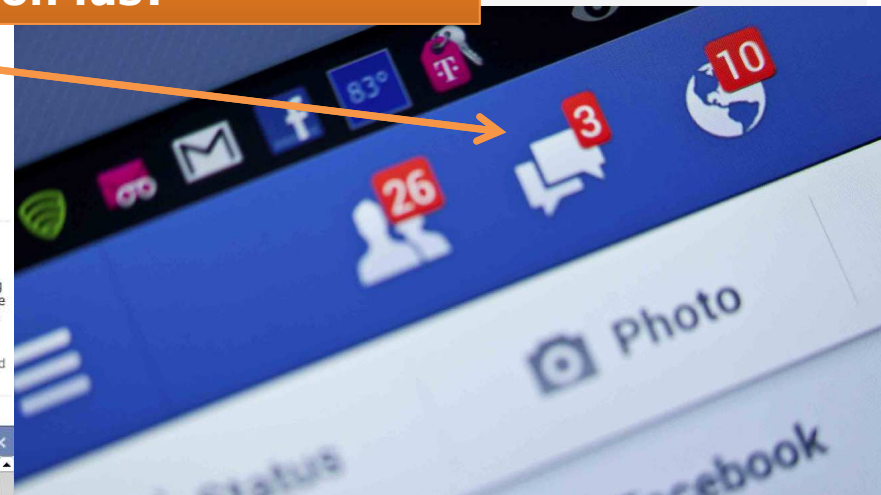
# Limites de MVC

- En 2014, **Facebook** a annoncé son passage au nouveau modèle de programmation **Flux** développé par ses soins, délaissant par la même occasion le modèle MVC.
- MVC n'étant plus adapté aux besoins de l'immense quantité de données transitant à travers Facebook.
- La complexité du modèle MVC se faisait ressentir à chaque introduction d'une nouvelle fonctionnalité.
- Les ingénieurs craignaient de plus en plus de comportements hasardeux de l'ensemble.

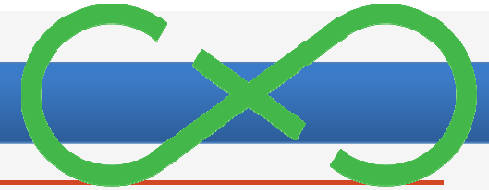


# Limites de MVC

Comment garder la cohérence de l'état (State) sur la notification des messages non lus?







- **Flux :**

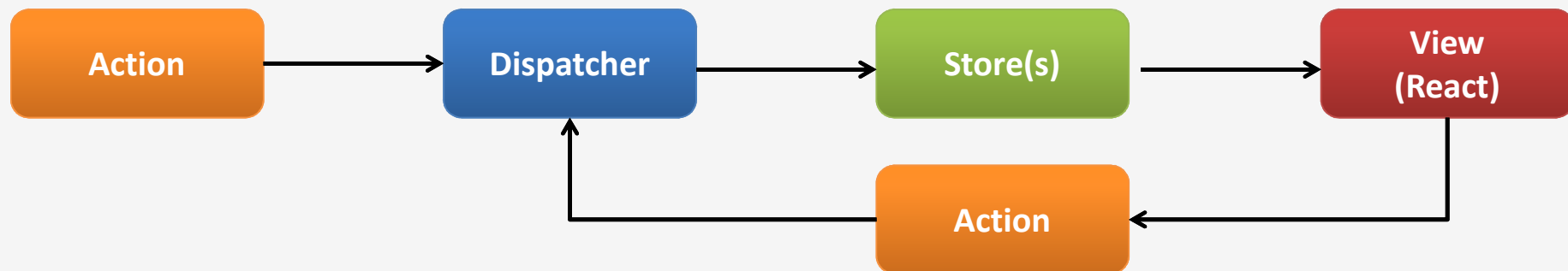
- Créé par facebook, Flux est une architecture permettant de gérer les données des applications.
- Il se caractérise par le fait que le flux de données est **unidirectionnel**.
- C'est ce que l'on appelle en architecture fonctionnelle le "**one-way data flow**", le flux de données va toujours dans le même sens
- En comparant (React + Flux) à un design pattern en MVC (Model-View-Controller) :
  - Flux correspondrait au M (Model) de MVC,
  - React prendrait le rôle du V (View).

# Architecture Flux



- **Composantes de Flux :**

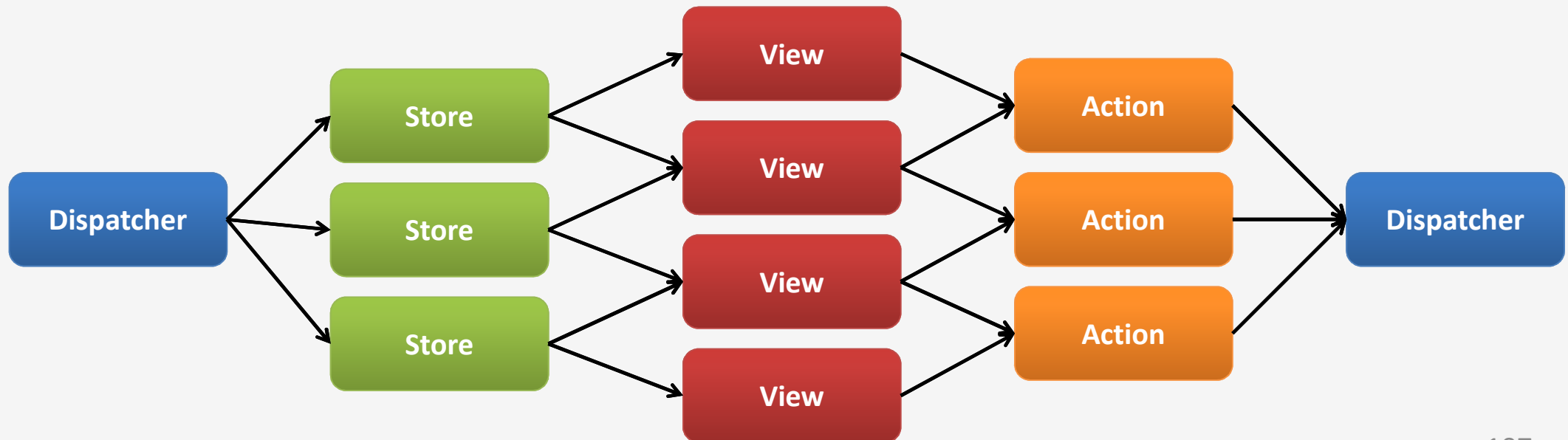
- **Store(s)** : un ou plusieurs stores contenant toutes les données de l'application
- **Action** : flux d'information que l'on souhaite envoyer au state global
- **Dispatcher** : remplace le Controller initial en décidant comment le Store doit être mis à jour lorsqu'une Action est déclenchée.
- **View** : écoute les modifications de store pour se mettre à jour éventuellement en générant une Action pour traitement par le Dispatcher.



# Flux : Multiples Stores ou Views



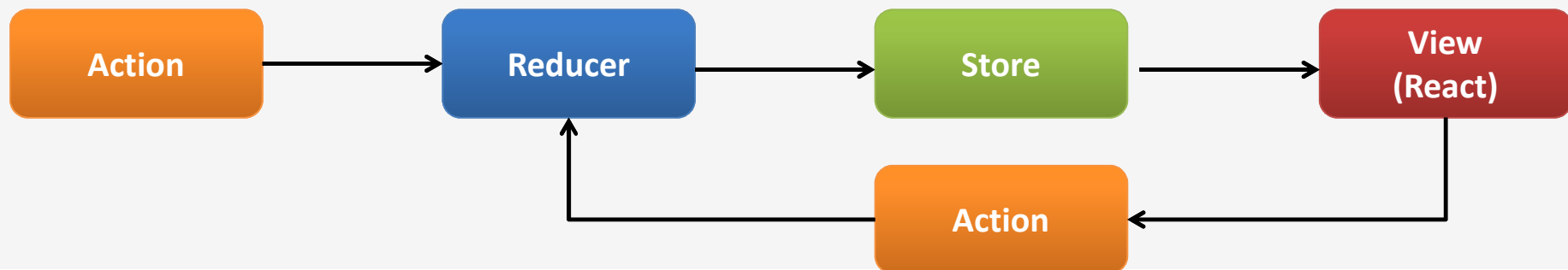
- Un système avec de multiples *Stores* ou *Views* peut être vu comme n'ayant qu'un seul *Store* et qu'une seule *View* puisque les données ne s'écoulent que dans un sens et que les différents *Stores* et *Views* n'interagissent pas entre eux directement.
- Ceci garantit un flux unidirectionnel des données à travers les composants (views) du système.



# Redux : c'est quoi ?



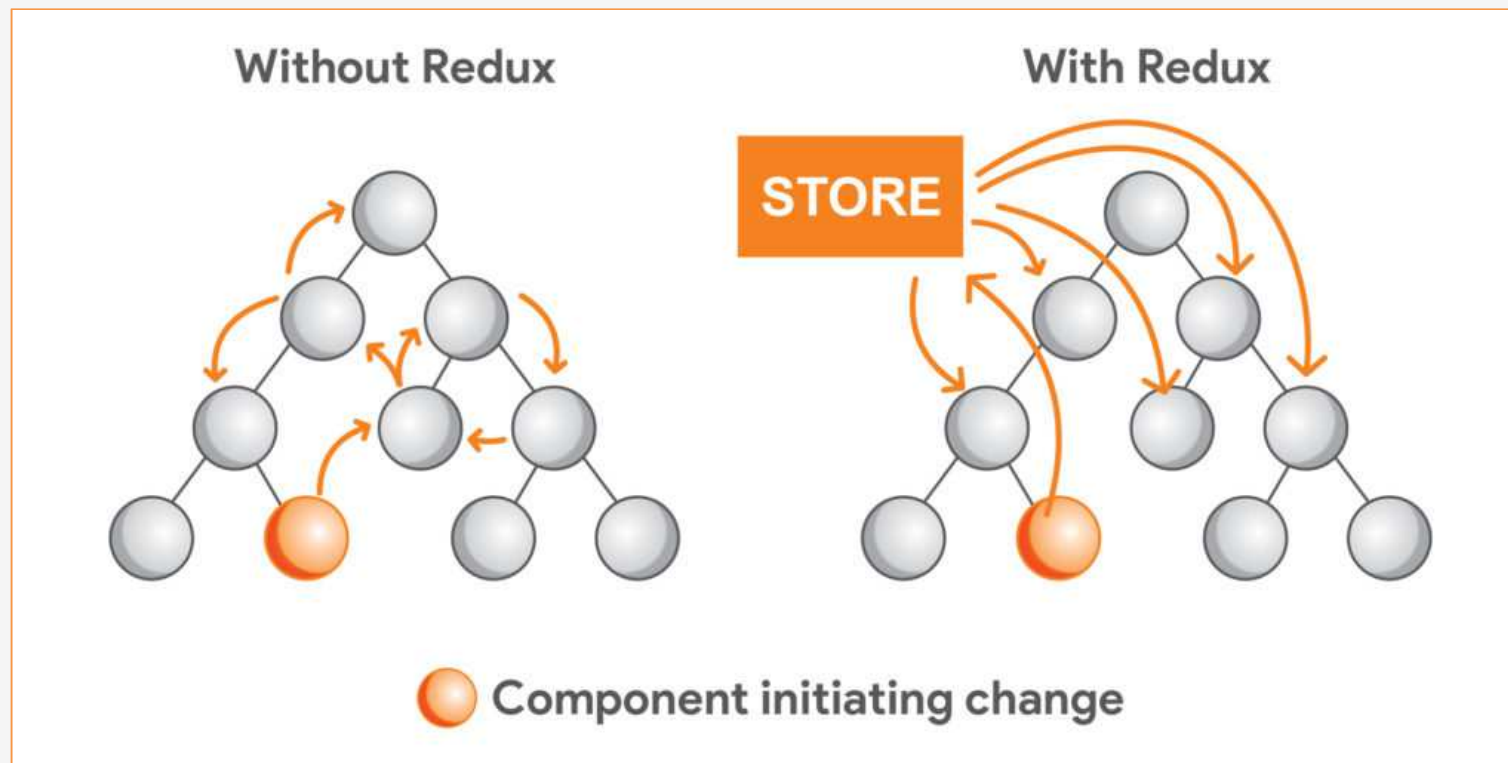
- **Redux** : librairie JavaScript, développée par Dan Abramov depuis mai 2015.
- Il est aussi possible d'utiliser Redux avec d'autres Frameworks comme Angular ou Ember.
- Il s'agit d'une évolution de Flux qui reprend ses principales caractéristiques : flux unidirectionnel, store émettant des notifications auprès des composants. Mais d'une manière tout à fait différente.
- L'architecture Redux introduit des nouveaux composants à savoir :
- **Reducer** : on peut assimiler à une sorte de contrôleur (MVC) attendant le déclenchement d'actions.
- **Store** : Un unique store centralisé et non mutable correspondant au modèle (MVC).
  - **Redux = Flux + Reducer**



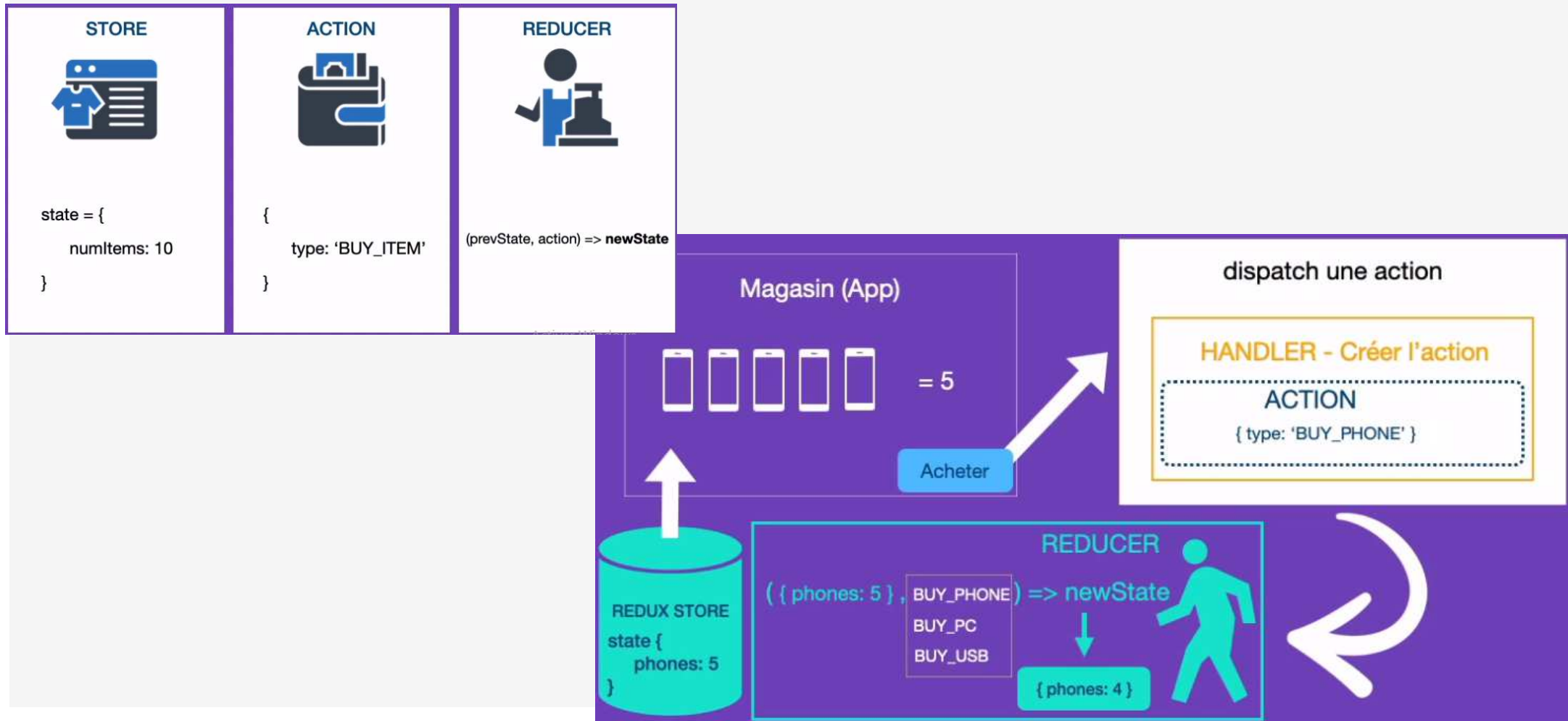
# Redux



- Pourquoi Redux ?



# Redux : principe



# Redux : 3 concepts clés (1/3)



- **Store de redux : Global state**

- Avec Redux, tout l'état d'une application est décrit comme un objet.
- **Le global state est unique :**
- le global state doit être constitué d'un arbre unique et stocké dans un unique store.
- Par exemple, pour une application qui recenserait des tâches à faire (todos), celle-ci pourrait ressembler à :

- **Exemple du store :**

```
{
  todos: [
    {
      id: 1,
      text: 'Apprendre à utiliser Redux.',
      cat: 'learn',
      completed: false,
    },
    {
      id: 2,
      text: 'Apprendre à utiliser React.',
      cat: 'learn',
      completed: true,
    }
  ],
  visibilityFilter: 'SHOW_ALL'
}
```

## Redux : 3 concepts clés (2/3)



- **Actions**

- Pour modifier l'état de l'application, il faut utiliser une **action**.
- **Le state n'est qu'en lecture seule :** la seule façon de modifier un state est d'appeler une action qui appliquera la logique définie dans les **reducers**.  
Par conséquent, le state ne doit jamais être modifié directement.

- **Exemples d'actions :**

```
{  
  "type": "ADD_TODO",  
  "text": "Apprendre à intégrer une API",  
  "cat": "learn"  
}  
  
{  
  type: 'EDIT_TODO_CAT',  
  id: 2,  
  cat: 'best practice'  
}  
  
{  
  type: 'SET_VISIBILITY_FILTER',  
  filter: 'SHOW_ALL'  
}
```



# Redux : 3 concepts clés (3/3)



- **Reducers**

- Pour relier les state du global state défini précédemment aux actions, on écrit ce que l'on appelle des **reducers** : ce sont des fonctions qui prennent en arguments un **state** et une **action**, et qui retournent le **state** après que l'action ait été effectuée.

**(state, action) => NewState**

- **Les reducers sont des fonctions pures :**

c'est-à-dire qu'elles ne peuvent pas prendre d'arguments extérieurs à elles-mêmes.

```
function visibilityFilterReducer (state = 'SHOW_ALL', action) {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return action.filter
    default:
      return state
  }
}

function todosReducer (state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [ ...state, {
        id: state.reduce((maxId, todo) => Math.max(todo.id, maxId), -1) + 1,
        text: action.text,
        cat: action.cat || 'default',
        completed: false,
      } ]
    case 'EDIT_TODO_CAT':
      return state.map( (todo, id) => action.id === id
        ? { ...todo, cat: todo.cat }
        : todo )
    default:
      return state
  }
}
```

# La méthode combineReducers()



- **rootReducer :**

Sorte de fichier de configuration pour combiner les valeurs retournées par les **reducers** aux différentes variables qui vont être définies au niveau du **store** :

- Clé : nom de variable (pour définir chaque bloc de données du store)
- Valeur : valeur de retour du reducer.

- **combineReducers() :**

- englober tous les reducers dans un "root reducer", qu'on utilisera lorsqu'on appellera la méthode **createStore**

```
import {combineReducers} from 'redux'

const rootReducer = combineReducers({
  todos = todosReducer;
  visibilityFilter = visibilityFilterReducer;
});

export default rootReducer;
```

Le store

```
{
  todos: [
    {
      ...
    },
    {
      ...
    }
  ],
  visibilityFilter: ...
}
```

Les reducers

```
function
visibilityFilterReducer(state = 'SHOW_ALL', action){
  ...
}
function todosReducer (state = [], action) {
  ...
}
```

# Inclure le store dans React



- Pour créer un store, redux utilise la méthode : ***createStore***
- createStore prend en paramètre le(s) reducer(s) : ***createStore(reducer)***
- Pour donner accès au store aux différents composants d'une application React, Redux offre le composant ***Provider***
- ***Provider*** : distribuer le store à toute l'application.

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

ReactDOM.render(
  <Provider store={store}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Provider>,
  document.getElementById('root')
);
```

```
import { createStore } from "redux"

var store = createStore(rootReducer){
  return state;
};
```

# Redux : l'objet store



L'objet **store** possède trois méthodes :

- **getState** : retourne l'état courant du store. L'objet retourné ne doit pas être modifié.
- **subscribe** : 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 modification et mettre à jour l'interface graphique en conséquence.
- **dispatch** : prend en paramètre une action et exécute le reducer qui va, à son tour, mettre à jour le store avec un nouvel état.

```
store.subscribe(() =>
  console.log(store.getState()))
// 0

store.dispatch({
  type: 'INCREMENT' })
// 1

store.dispatch({
  type: 'INCREMENT' })
// 2

store.dispatch({
  type: 'DECREMENT' })
// 1
```

# Connecter un composant React au store Redux



- **connect()** : une fonction qui retourne une fonction d'ordre supérieur et reçoit en entrée un composant. Il nous aide à connecter notre composant au store et à nous donner accès à l'état.
- **mapStateToProps** : fournir les données du magasin au composant : injecter le **state** renvoyé par Redux dans les props de React
- **mapDispatchToProps** : Il connecte les **actions** de redux aux propriétés (props)

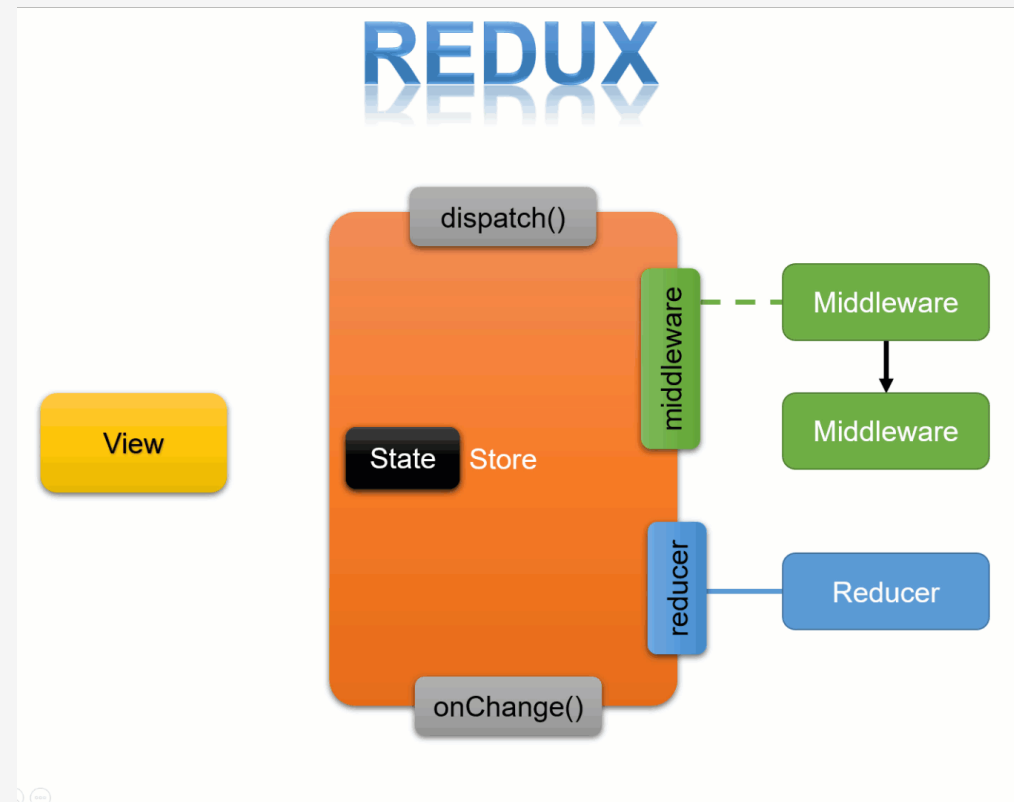
```
class monComposant extends Component {  
  afficher = () => {  
    this.props.afficher()  
  }  
  render() {  
    <div>  
      <h1>Le message est :  
        {this.props.message}  
      </h1>  
    </div>  
  }  
}
```

```
function mapDispatchToProps(dispatch) {  
  return({ afficher : () =>  
    {dispatch(ALERT_ACTION)}  
  })  
}  
function mapStateToProps(etat) {  
  return({message : etat.monMessage})  
}  
  
export default connect(mapStateToProps, mapDispatchToProps)  
(monComposant)
```

# Redux Middleware



- Un middleware permet de faire tourner du code après un dispatch mais avant qu'elle atteigne le reducer.
- Les middlewares peuvent être chaînés.
- Permet d'inspecter les actions, les modifier, les stopper, en déclencher d'autres...
- Exemple : redux-thunk
  - middleware qui aide à gérer le code asynchrone



# Redux : Avantages et Inconvénients



## Avantages

- Structure commune, quel que soit le projet;
- State centralisé, pratique pour le debug;
- Offre de la modularité pour les plus gros projets;
- Le code Redux est facilement réutilisable pour créer une UI alternative.



## Inconvénients

- Structure imposante ;
- Nécessité de décrire le state comme objet simple / tableau ;
- Nécessité de décrire le changement de state puis d'UI dans le code;
- Redux peut engendrer de légères pertes de performance.

## Récap. : MVC vs Flux vs Redux

	MVC	Flux	Redux
<b>Architecture</b>	MVC - Model View Controller: Modèle de conception architecturale pour développer l'UI	Architecture d'application conçue pour créer des applications Web côté client.	Bibliothèque JavaScript open source utilisée pour créer l'UI.
<b>Direction du flux de données</b>	Flux bidirectionnel	Flux unidirectionnel	Flux unidirectionnel
<b>Store uniques ou multiples</b>	Pas de concept de store	Comprend plusieurs stores	Comprend un simple store
<b>Où peut-on utiliser?</b>	Frameworks Frontend et Backend	Frameworks Frontend	Frameworks Frontend



# Redux : Installation



- **Redux :**

- > `npm install redux`

- **Packages complémentaires :**

- > `npm install react-redux`

- > `npm install --save-dev redux-devtools`

- Ou encore `npm install --save redux-devtools-extension`

- **Création d'une application React avec template redux**

- > `npx create-react-app my-app --template redux`

```
{ package.json > ...
1  {
2    "name": "my-react-redux-app",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.14.1",
7      "@testing-library/react": "^11.2.7",
8      "@testing-library/user-event": "^12.8.3",
9      "react": "^17.0.2",
10     "react-dom": "^17.0.2",
11     "react-redux": "^7.2.4",
12     "react-scripts": "4.0.3",
13     "redux": "^4.1.1",
14     "web-vitals": "^1.1.2"
15   },
```

- **Atelier 4 :**
  - ❑ Développement d'une application SPA avec ReactJS et Redux

# Merci

**Looking for a training,  
an internship or want  
to attend an event?**

16000 young Tunisians have already been trained and supported in digital, 1800 have benefited from professional conversion courses, 800 middle and high school students have been introduced to coding, and 95% of them have already found jobs in Tunisia and abroad.



Visit us at  
**innovation.orange.tn**



Like us on Facebook  
**OrangeTN.plus**



Follow us on Twitter  
**@OrangeTN\_Plus**