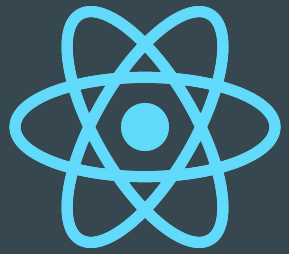


React JS - Introduction

Romain Jalabert - École IPSSI

React - Contexte

React - Quid ?

React est une librairie **Javascript** orientée front permettant de construire des interfaces utilisateurs. Ce langage fut initialement développé par Facebook (enfin “Meta” maintenant) et devint accessible au grand public en **2013**.

Le réseau social ayant des données changeantes en permanence, il fallait trouver un langage qui permette à la vue de réagir (d'où le nom) et de changer selon les interactions utilisateurs et les événements.

Disclaimer : React est un langage relativement accessible mais peut s’avérer problématique si les fondamentaux de Javascript ne sont pas bien maîtrisés.

React - Pourquoi ?

- Mieux synchroniser ce que l'on appelle "l'état" et la "vue", en faisant de la vue une fonction de l'état. C'est à dire que lors du changement d'état, le changement de vue adapté est déclenché.

- Afin de construire des apps basées sur une logique de composants

- Pour profiter des très nombreuses librairies et packages disponible en croissance constante

**on va y revenir*

Exemples de sites et app codés en React

- Facebook
- Instagram
- Air bnb
- Uber Eats
- Netflix
- Atlassian Task Management
- Dropbox
- Paypal

React JS seule option ?

Si React est encore massivement populaire, le débat autour du “meilleur” framework/library front reste entier. De nombreux développeurs vont notamment préférer Vue ou Angular (et autres Ruby on rails).

À titre d’exemple, React est parfois critiqué à cause des nombreuses dépendances souvent nécessaires au bon fonctionnement d’un projet complexe, ce qui peut rendre l’app lourde et entraver certains aspects de sa performance.

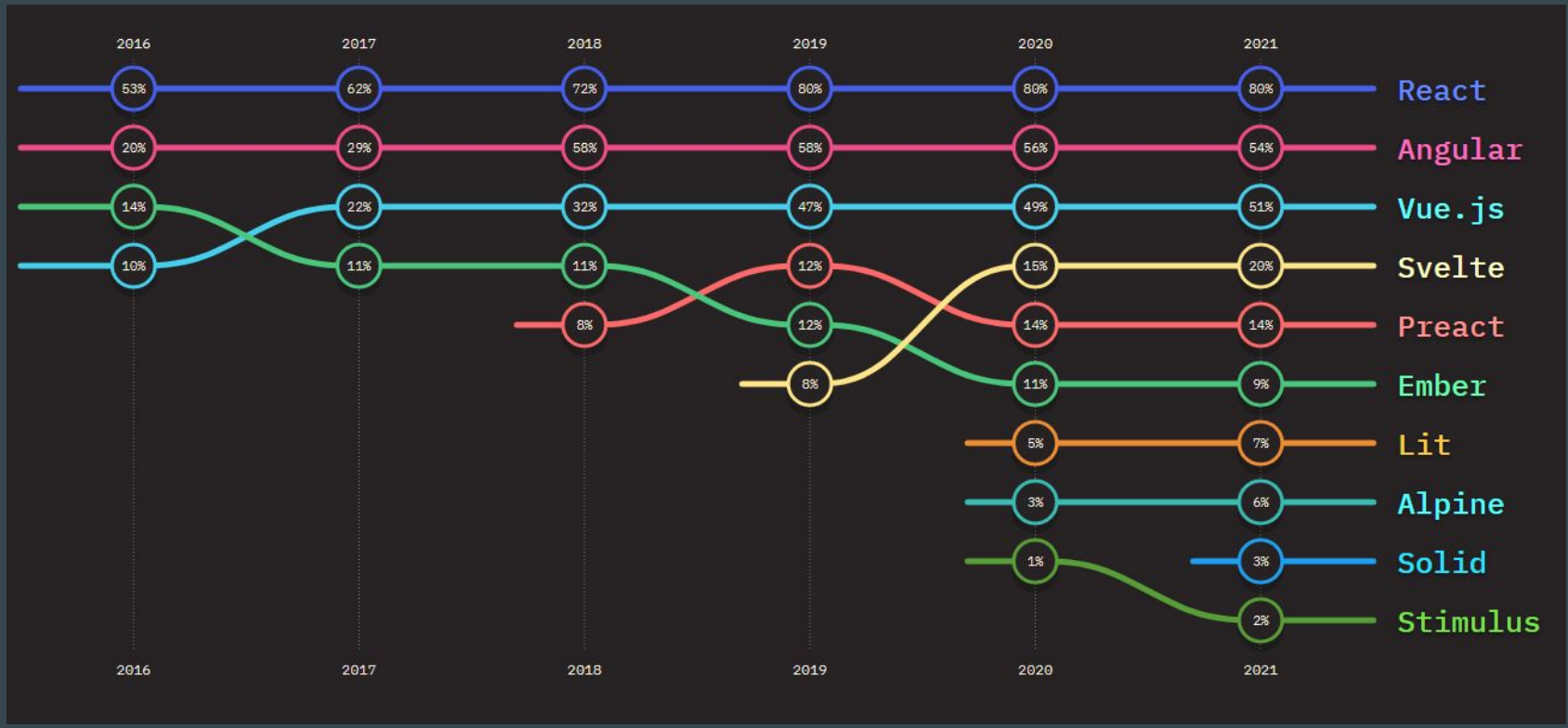
Angular

Vue

Svelte

Next JS

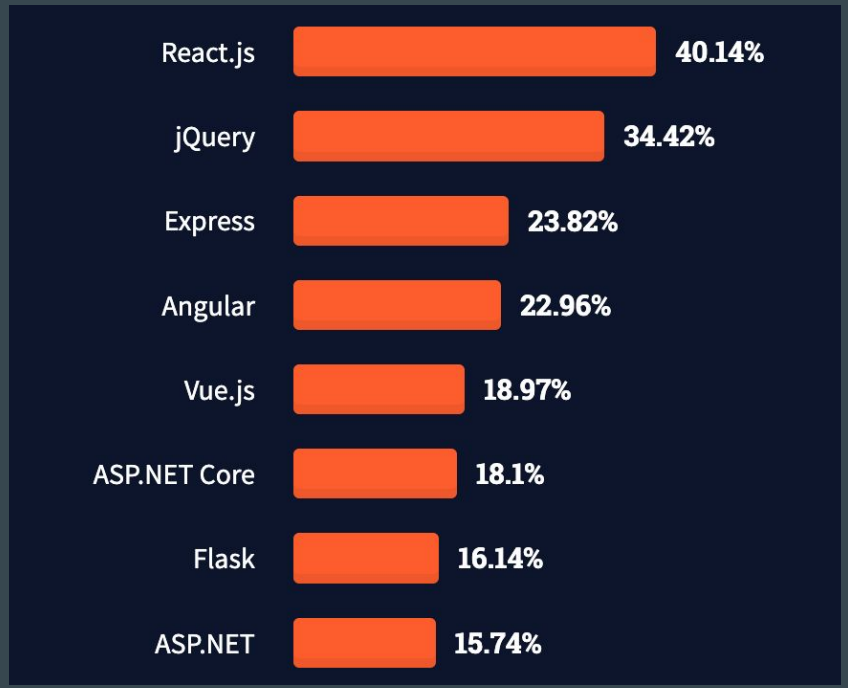
JS annual state survey 2021

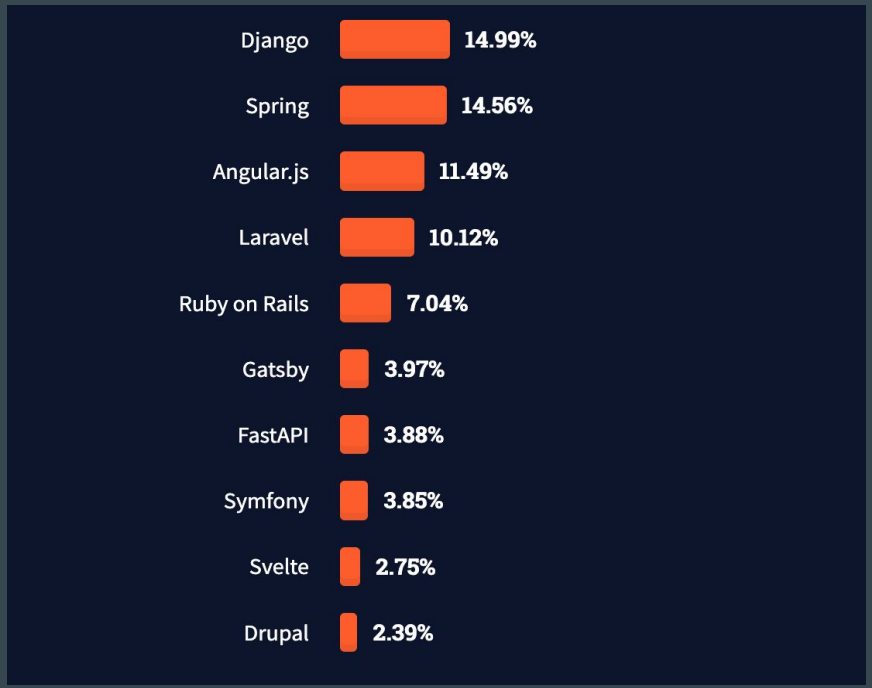


source : Annual state of javascript survey : https://2021.stateofjs.com/en-US/

Stack Overflow survey 2021

Which web frameworks and libraries have you done extensive development work in over the past year, and which do you want to work in over the next year? (67 593 answers)





source : https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks

REACT **JS** : ES6 features

ES5+ ES6+ / Babel

ES6 signifie **ECMAScript 6** , cela correspond à la version 6 de ECMAScript à savoir une version standardisée de Javascript avec de nombreuses features :

- const et let
- Arrow functions
- Method.map
- Classes
- Les modules
- Le spread operator

Dans le contexte de React on transpile l'ES6 en ES5 via Babel afin qu'un maximum de navigateurs reconnaissent le langage et puissent l'exécuter, l'ES5 étant une version reconnue par la plupart.

ES6+ : var, const et let

var : variables qui peuvent être re-déclarées et mise à jour. Portée globale lorsque déclarée hors d'une fonction, portée dans la fonction seulement lorsque la variable est déclarée à l'intérieur de celle-ci.

const : variable à assignation unique, c'est-à-dire dont la valeur est figée une fois établie. Elle ne peut donc pas être déclarée à nouveau. Sa portée est limitée au bloc* dans lequel elle est déclarée.

let : Peut être mise à jour mais pas re-déclarée. Sa portée est limitée au bloc dans lequel elle est déclarée.

*Un bloc = ce qui est entre {}

Si besoin de précisions :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/const

ES6+ : Arrow Functions / Fonctions fléchées

Fonction handleClick en ES5 :

```
function handleClick() {
   console.log('click')
}
```

Adaptée en ES6 (fonction fléchée) :

```
const handleClick = () => {
       console.log('click')
   }
```

Les fonctions fléchées en React sont une façon plus concise d'écrire des fonctions. Elles sont souvent utilisées pour définir des composants fonctionnels dans React.

À noter que l'utilisation d'accolades après la flèche d'une fonction fléchée en React n'est pas obligatoire si la fonction ne retourne qu'une seule expression. Si la fonction ne comporte qu'une seule instruction, la valeur de cette instruction est automatiquement retournée.

LET'S GO !

Premières étapes - Node JS

Se placer dans le bon dossier. En créer un si besoin :

```
mkdir my-folder
```

S'assurer dans un premier temps que Node JS (https://nodejs.org/en/download/) est bien installé :

```
node -v
```

sinon : `brew install node`

Mettre à jour Node :

```
sudo npm install -g n
sudo n stable
```

Package Manager

Bien s'assurer que vous avez installé et mis à jour votre Package Manager.

Celui-ci nous permettra d'installer notamment les packages et dépendances souhaitées (ex : react, scss, babel)

Il y a 2 package managers connus, NPM et Yarn :

NPM (Node Package Manager)

Vérifier la version : `npm -v`

Installer la dernière version : `npm install -g npm`

YARN

Vérifier la version : `yarn -v`

Installer yarn : `npm install --global yarn`

Premières étapes - Create React App

React nécessite pour son bon fonctionnement plusieurs **packages** lors de son installation. Cela peut vite devenir fastidieux si il faut s'en occuper à chaque que l'on démarre un projet. Le package initial, ou "Boilerplate", de React comprend :

React (jusqu'ici tout va bien)

Webpack -> Compile l'ensemble des fichiers JS en un fichier (bundle)

ESLint -> Aide à écrire un meilleur code (en checkant les erreurs)

Babel -> Compile de l'ES6 en ES5

Installer individuellement chaque élément et les connecter peut prendre un (très) long moment ... Par conséquent afin d'éviter cela nous allons nous intéresser à la commande "create-react-app"

Premières étapes - Create React App

Le moyen le plus simple de commencer un projet React en épargnant les commodités d'installation :

```
npx create-react-app my-app
```

Se déplacer dans le bon dossier :

```
cd my-app
```

Lancer le script :

```
npm start
```

Alternative Vite !

Create-react-app est de plus en plus décrié chez les développeurs React.

La lourdeur du projet créée en est souvent la cause.

C'est pourquoi de nouvelles alternatives existent comme Vite. Cet outil permet de créer facilement un nouveau projet dans le langage de votre choix (react, vue, js etc)

Toutes les infos pour créer un projet via vite ici :
https://vitejs.dev/guide/

React - Basics

JSX

Le JSX, ou Javascript XML, est le langage spécifique de React, il sert à définir la vue de nos composants.

Attention cependant, le JSX est une syntaxe qui ressemble fortement à du HTML mais est en réalité du Javascript, par conséquent :

- Attention au nom des « attributs ». Par exemple : bien écrire ** ** car en JS c'est le propriété className qui définit la valeur de l'attribut HTML class.
- Les propriétés d' évènement (ex. <button onClick={func} />) s'utilisent de manières plus courantes qu'en HTML ou elles sont déconseillées.

Ci dessous un aperçu de la syntaxe :

```
const Ingredients = () => <div>
  <p>implémenter les ingrédients</p>
  <ul>...</ul>
</div>;
```

Render avec createRoot

Dans React on va vouloir afficher (‘render’) du HTML sur notre page Web avec la méthode ReactDOM.createRoot()

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

Disclaimer : C’était possible avec ReactDOM.render() ... Mais depuis la version 18 de React (Mars 2022) cette méthode est deprecated

https://beta.reactjs.org/reference/react-dom/client/createRoot

Virtual DOM

Qu'est ce que le DOM ?

Pour faire court, le **DOM (Document Object Model)** est une interface générée par le navigateur qui représente notre site avec ses différents nœuds sous forme d'arborescence.

Il y a deux aspects à considérer dans le DOM, d'une part la représentation du document HTML et d'autre part l'API qui manipule cet objet. Le DOM en 1998 n'a pas été créé pour les applications complexes développées aujourd'hui, **la manipulation de celui-ci de manière intense via l'API peut conduire à des problèmes de performance** .

Le **DOM virtuel** a été créé pour résoudre ce genre de problèmes. Contrairement au DOM, le DOM virtuel n'est pas une spécification officielle mais plutôt une nouvelle façon d'interagir avec le DOM.

Un DOM virtuel peut être vu **comme une copie du DOM original.** Cette copie peut être manipulée et mise à jour fréquemment, sans utiliser les API DOM. Une fois toutes les mises à jour effectuées dans le DOM virtuel, nous pouvons voir quels changements doivent être apportés au DOM original et les réaliser d'une façon ciblée et optimisée.

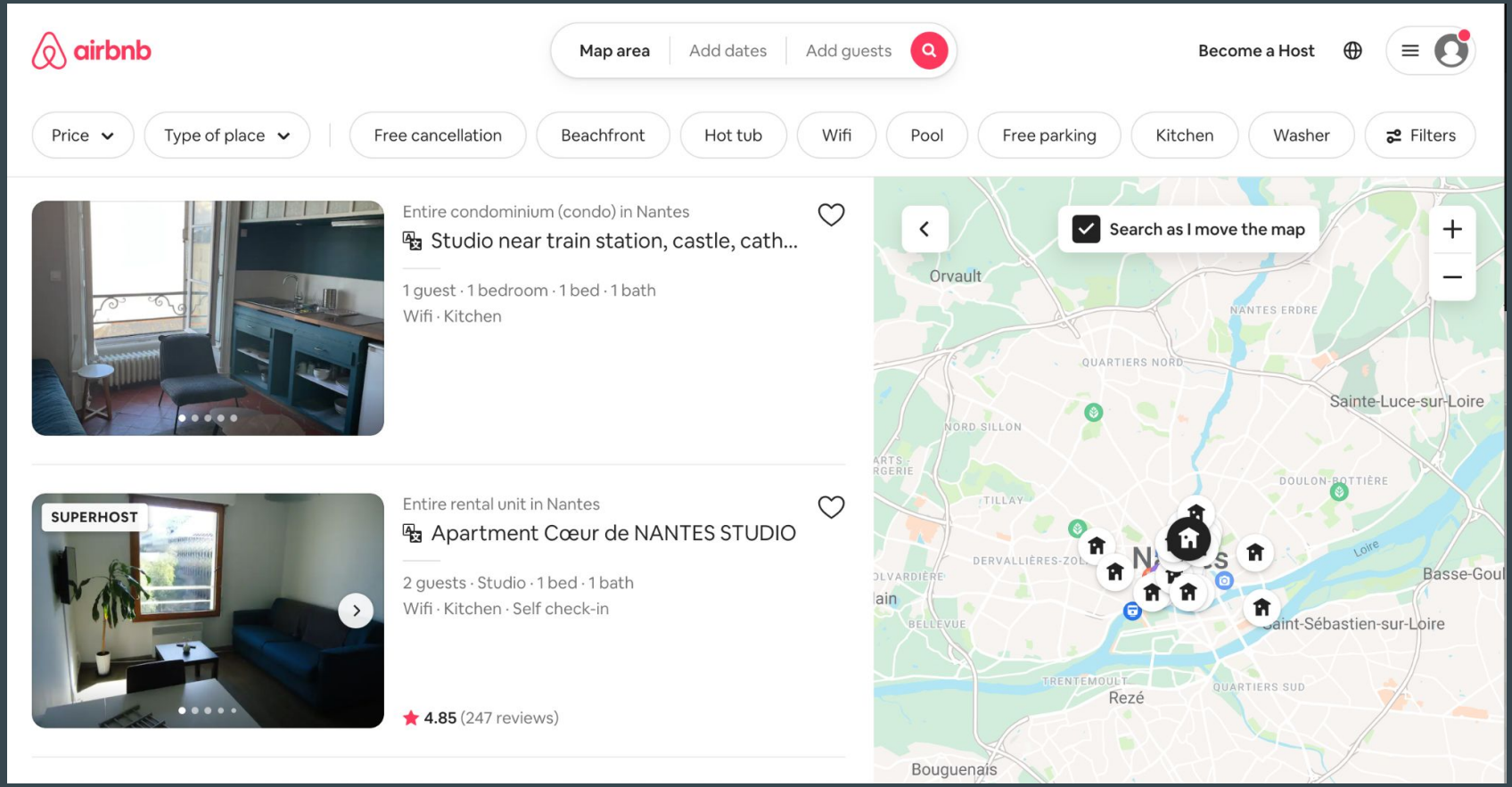
Components Logic

Un composant est une fonction qui retourne un élément React (en syntaxe JSX). Exemple classique de composant : header, navbar, modales, menu, etc.

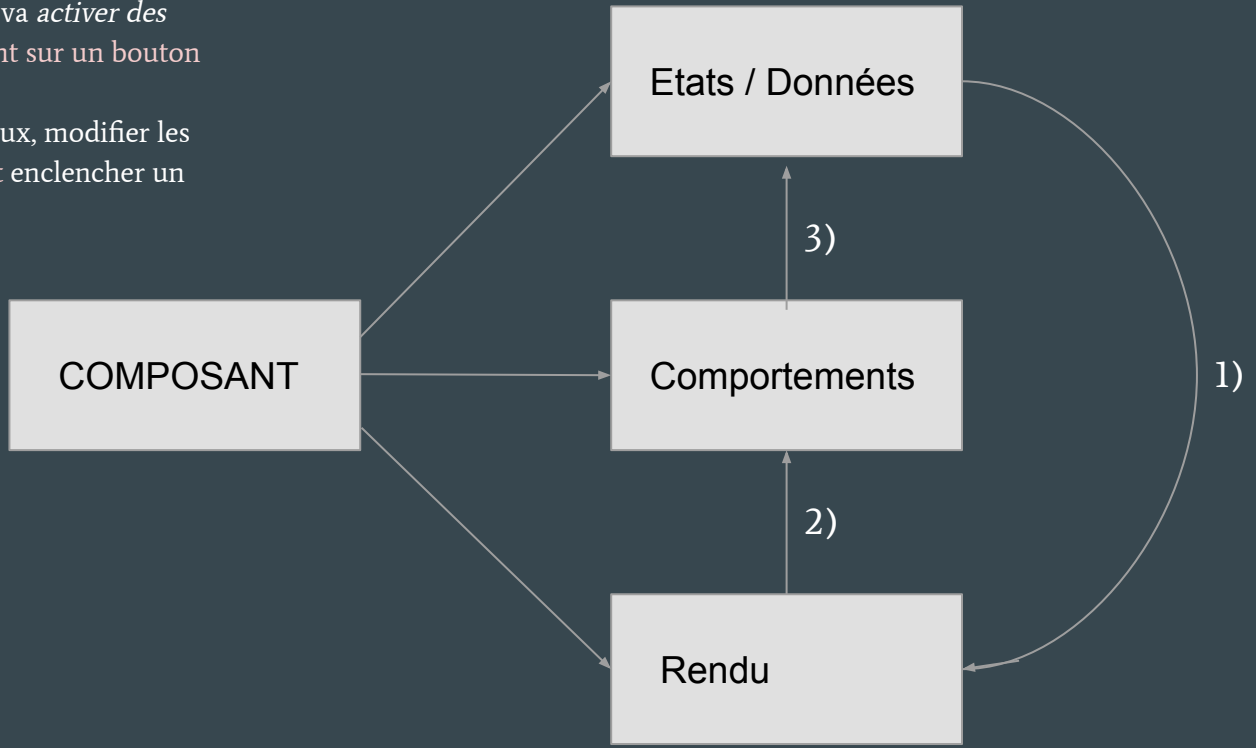
```
// Composant de présentation
function Header(props) {
  return <div id="app-header">
    <h1>{props.title}</h1>
    <span>Auteur : {props.author}</span>
  </div>;
}

export default Header;
```

Chaque composant créé doit être exporté (ce qui explique le export défaut dans notre exemple de gauche) et donc aussi importé dans le component parent (App par exemple).

Quels sont les composants identifiables sur la page suivante ?



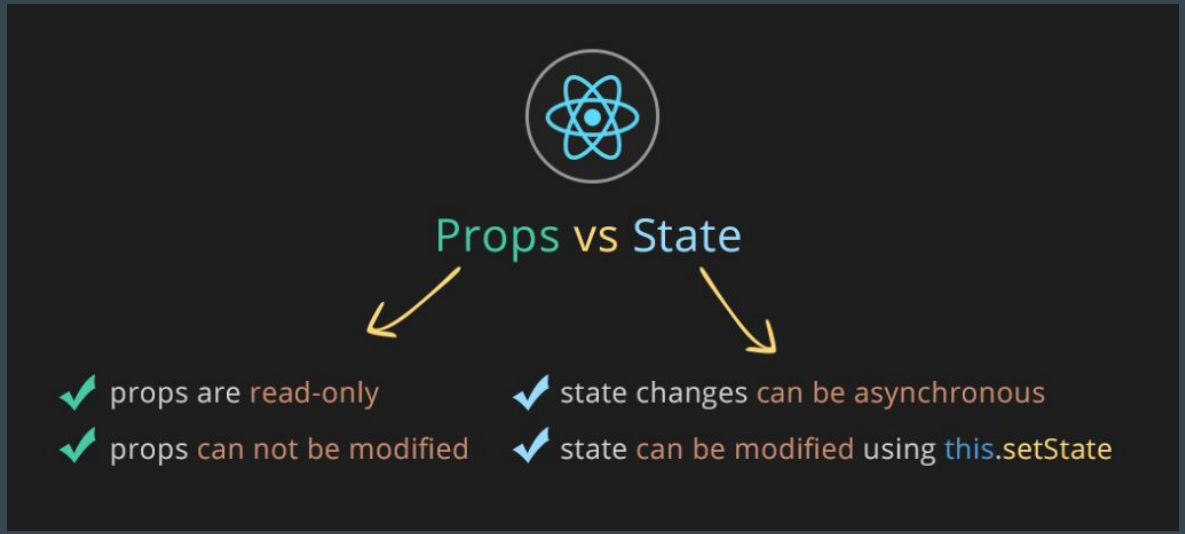
1) Le rendu affiche les données. Il est aussi dépendant de celles-ci car le changement d'états / données provoque automatiquement une actualisation du rendu.
2) Cependant via le rendu on va *activer des comportements* (en cliquant sur un bouton par exemple)
3) Comportements qui vont eux, modifier les données, et par conséquent enclencher un nouveau rendu.



Props and State

Props : C'est de l'information que l'on passe entre composants, qui pourrait se comparer aux arguments d'une fonction (params).

State : C'est aussi de l'information mais géré à l'intérieur même du composant et sujette à changement. Ce serait comparable à une variable déclarée au sein d'une fonction.



Props exemple

1) On ajoute l'attribut "brand" à notre composant "Car"

```
const myelement = <Car brand="Ford" />;
```

2) Le composant reçoit l'argument dans l'objet props via "props.brand"

```
function Car(props) {
  return <h2>I am a { props.brand }!</h2>;
}
```

Props résumé

- Pour passer des props on vient les ajouter au JSX comme on le ferait avec des attributs HTML
- Pour récupérer des props utiliser la syntaxe déstructurante : function `Avatar({ person, size })`
- On peut spécifier une valeur par défaut (ex: size = 100) qui sera utilisée pour des props manquantes ou undefined
- On peut passer toutes nos props vers un composant comme suit : `<Avatar {...props} />`
- Les props sont des captures d'informations à l'instant t. Elles sont read-only et à chaque fois qu'un composant render il reçoit une nouvelle version des props
- **On ne peut pas changer les props.** Si on souhaite créer de l'interactivité on utilise **setState**

Functional Components

Cependant en 2023 on utilise principalement des **composants fonctionnels** à savoir des composants constitués d'une fonction qui retourne directement du JSX.

Il est possible d'utiliser des composants fonctionnels grâce à l'existence des hooks (notamment pour gérer le state).

2 manière d'écrire des composants fonctionnels, exemple avec App :

Avec une Fonction

```
function Home() {
    return (
        <h1>Bienvenue sur mon app !</h1>
    )
}
```

Avec une Fonction Fléchée

```
Const = () => Home() {
    return (
        <h1>Bienvenue sur mon app !</h1>
    )
}
```

QUIZ 2

1) Quel est le langage utilisé lors de la construction de nos composants React ?

a) Javascript

b) HTML

c) JSX

d) Ruby

2) Comment appelle-t-on l'élément dont le changement déclenche une modification de la vue dans React ?

a) prop

b) param

c) state

d) boolean

3) Quelle est le sens du compilateur de code Babel

a) ES5 -> ES6

b) ES5 -> JSX

c) ES6 -> ES5

d) JSX -> ES6

4) Qu’est ce qu’un Hook dans react ?

a) Une manière de compiler du ES6

b) Un appel asynchrone à une API

c) Une fonction qui utilise l’état

d) Un fichier qui contient la liste des dépendances

4) Si j’ai un composant (avec bouton) de type compteur dont le nombre augmente quand je clique dessus. Cela signifie...

a) ... Que je viens de recevoir des props d’un autre composant

b) ... Que mon composant est “stateless”

c) ... Que mon état change à chaque click

Quelques références

Doc React - La doc officielle bien fournie

https://reactjs.org/docs/getting-started.html (ancienne en FR)
https://beta.reactjs.org/ (nouvelle doc mais en Anglais)

React Cheat Sheet - Tout le nécessaire syntaxique pour Réacter ! Toujours ouvert dans un onglet.

https://devhints.io/react

React Libraries - Ce serait dommage de pas en profiter !

https://reactlibraries.com/

Mozilla Developer Doc - Une référence mais pour le HTML, CSS, JS entre autres (pas de React malheureusement)

https://developer.mozilla.org/fr/docs/Web

Projets React

1) Créer une première app “My App” avec vite
2) Créer un premier composant App
3) Créer un composant Student dans App qui retourne un h1 ‘Bienvenue !’
4) Passer une prop “name” avec comme valeur votre nom à Student. Afficher le contenu de cette prop dans le h1. Ex : Bienvenue ! Je suis {prop}
5) Créer un deuxième composant Blog avec un h1 qui dit ‘Mon Blog’ aussi dans App
6) Passer des props à Blog (les importer depuis posts.jsx puis les passer)
7) Lister les articles et les afficher dans Blog (avec **map()**)