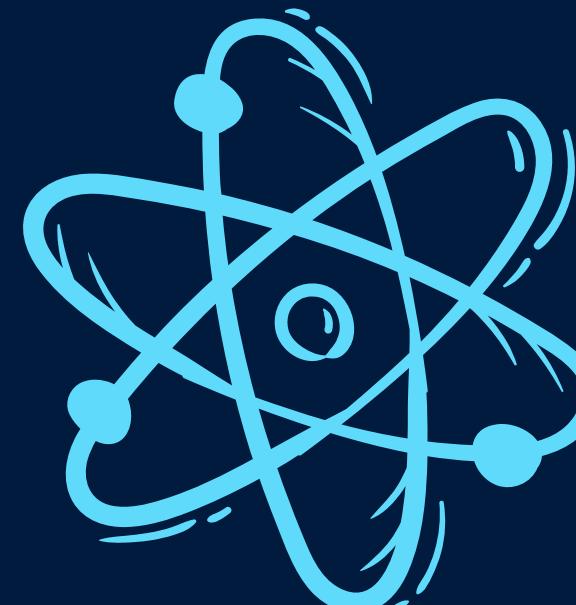




sonatel
ACADEMY

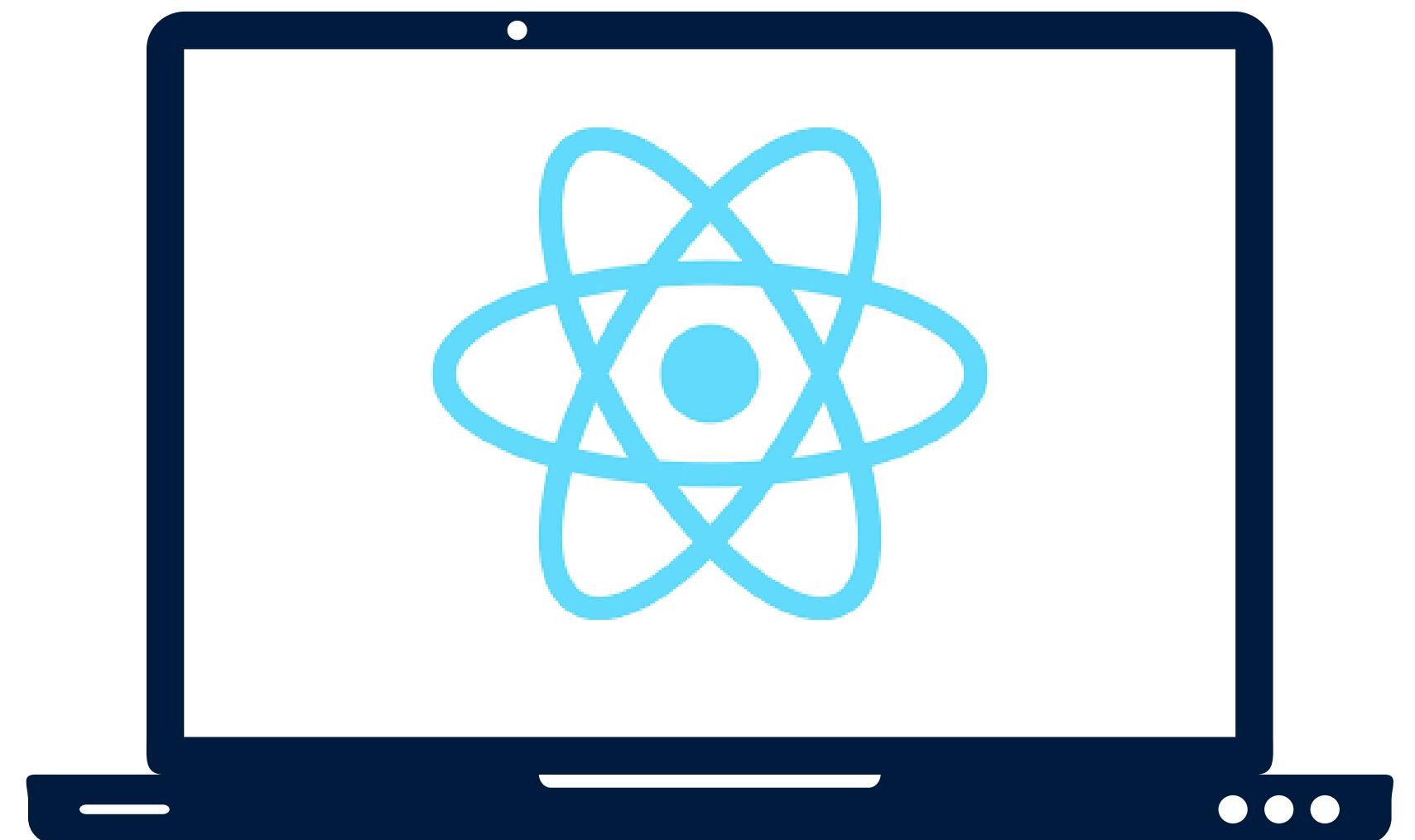
Présentation



@Groupe n°4

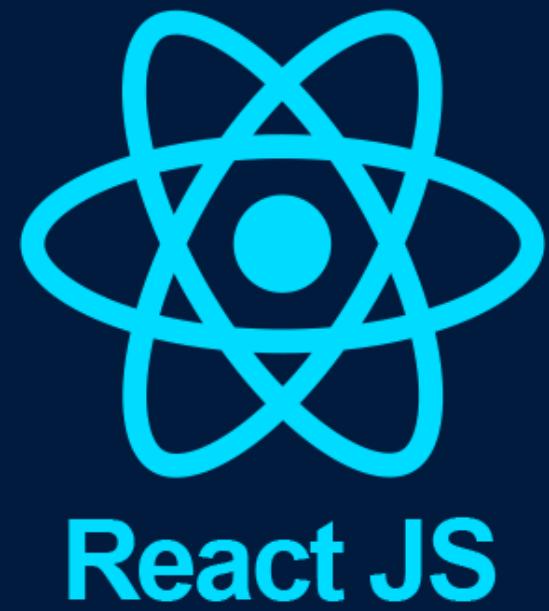
React JS

1





sonatel
ACADEMY



Membres du Groupe n°4



**Thiendo Ismaila
Mane**



Aicha Ndongo



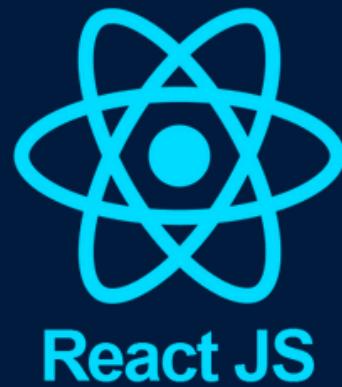
Aminata Seck



Ousseynou Ba

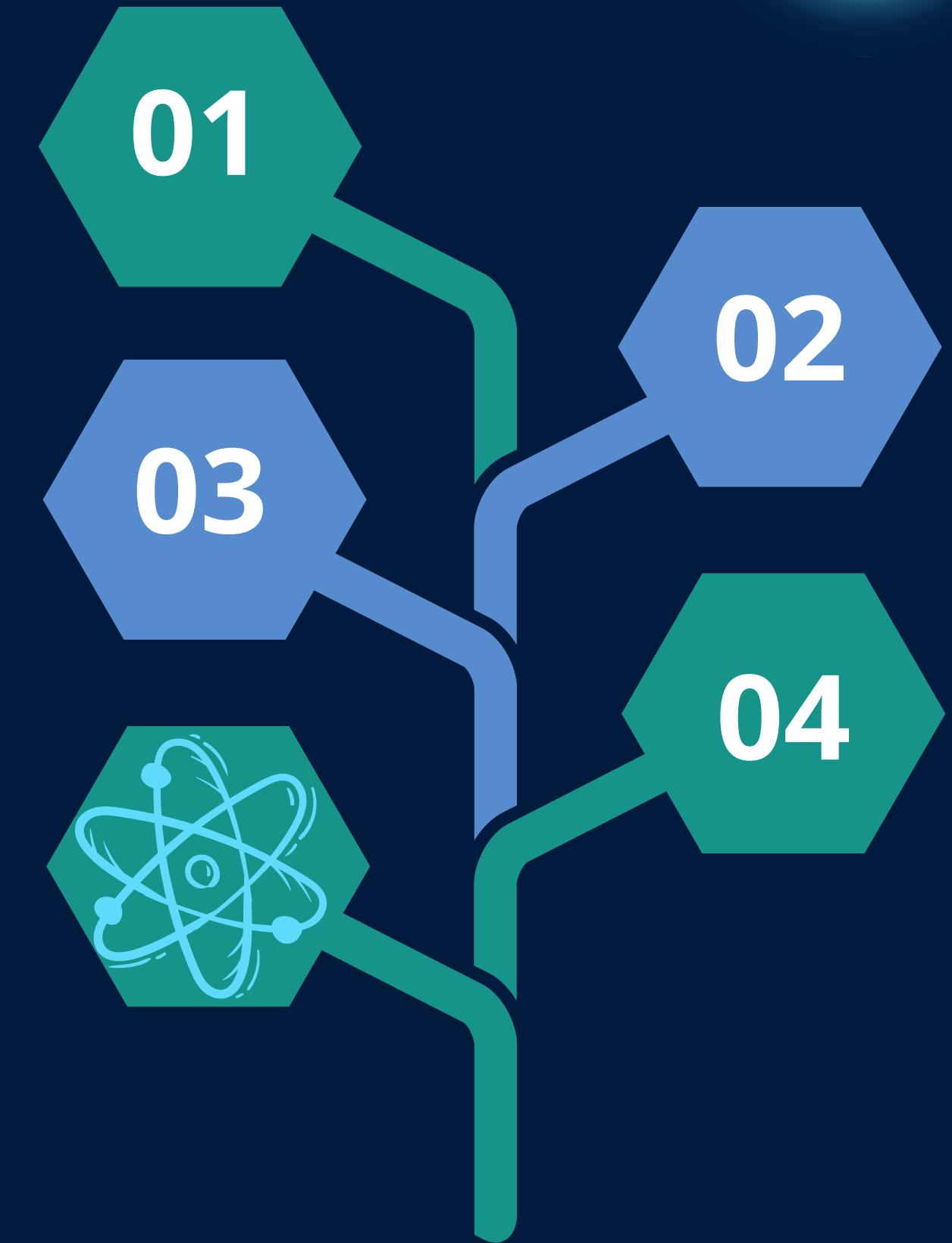


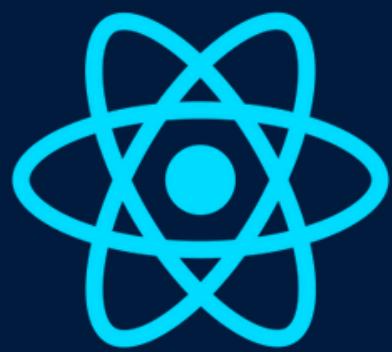
**Omar Louis
KAO**



SOMMAIRE

- 01 Présentation
- 02 Concepts
- 03 Démonstration
- 04 Conclusion



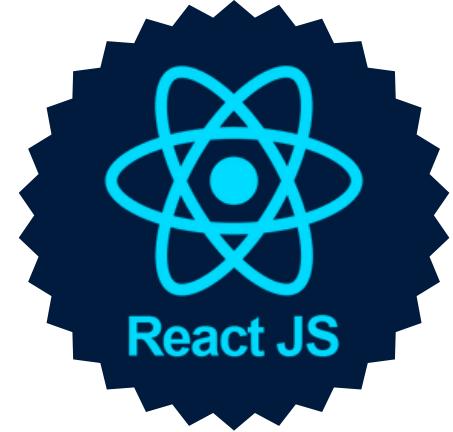


React JS



PRESENTATION

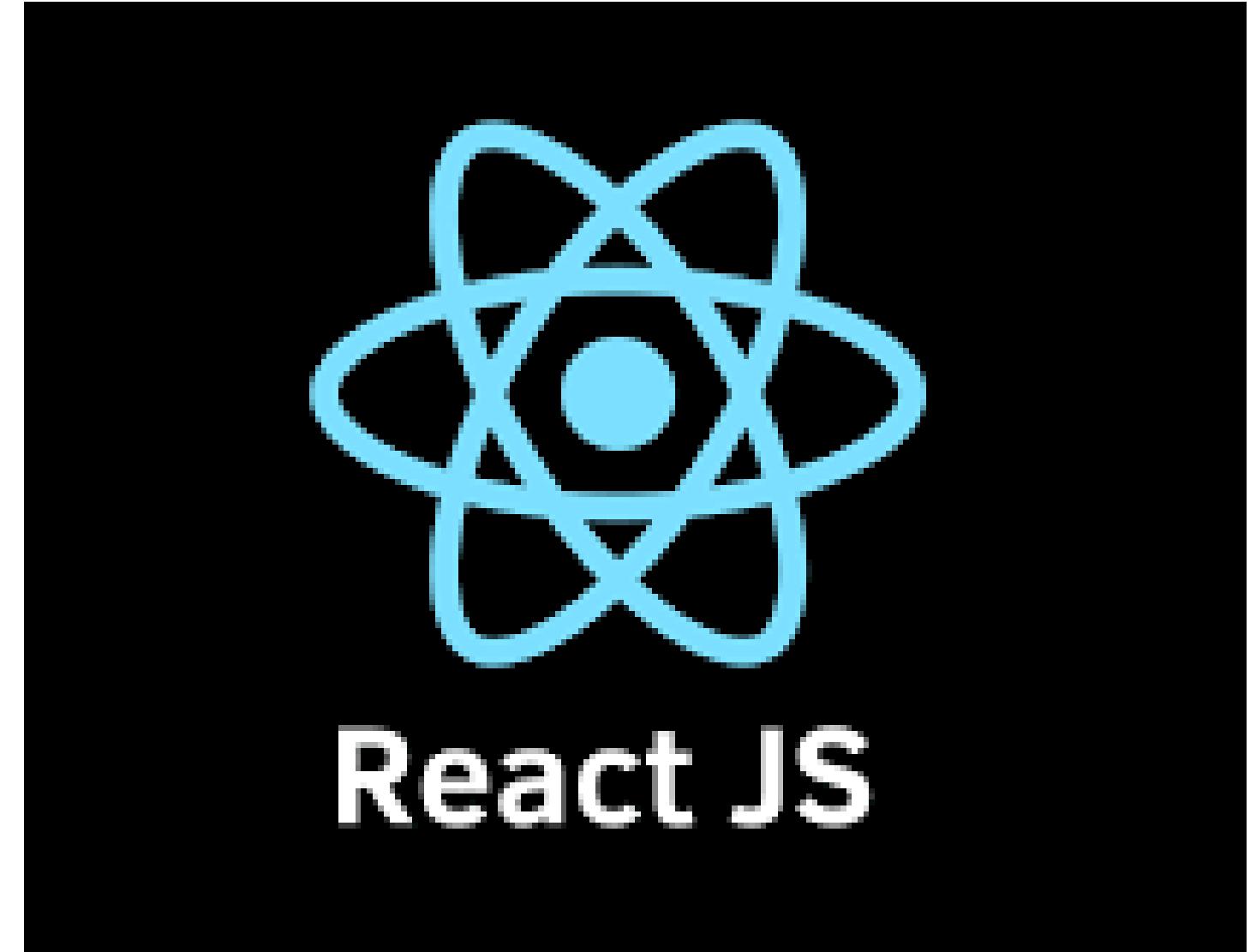


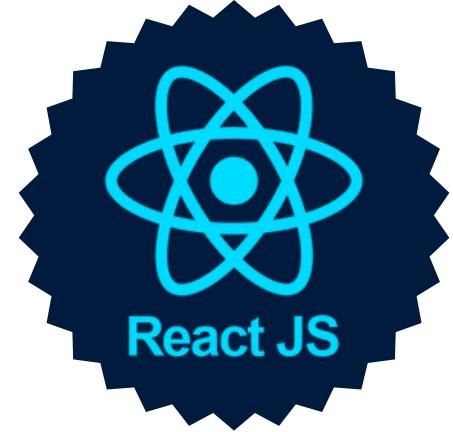


Qu'est-ce que React JS ?

5

- Une bibliothèque JavaScript
- Open-source
- Facebook, 2013
- UI/UX
- Approche modulaire





Pourquoi React ?

6

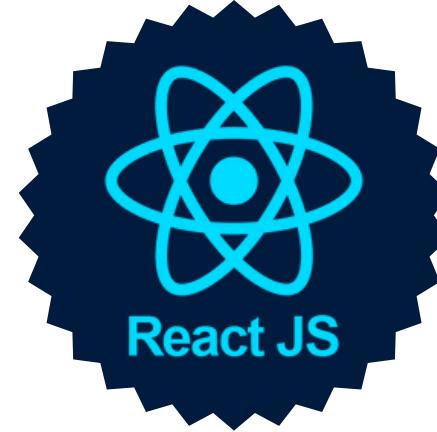
Avant React !?

- **jQuery**
- **AngularJS**
- **Backbone.js**



ANGULARJS

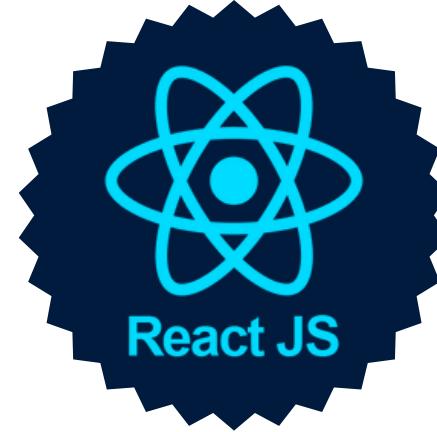




Les objectifs de React

7

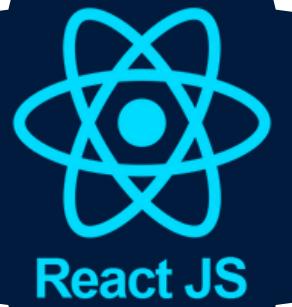
- **Simplification et organisation du développement**
- **Amélioration des performances des applications web.**
- **Facilitation la réutilisation du code**



Les innovations de React

8

- **Le Virtual DOM**
- **Une architecture basée sur les composants**
- **Un flux de données clair**



Installation de React

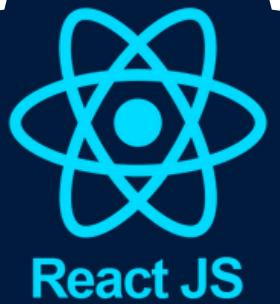
9

- **Etape 1: Installation de Node.js et npm**
- **<https://nodejs.org/en/download/>**

```
C:\Users\user>node -v  
v22.18.0
```

```
C:\Users\user>npm -v  
10.9.3
```

```
C:\Users\user>
```



React JS

Installation de React

10

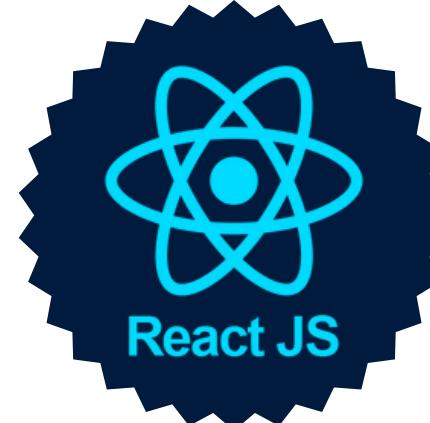
- **Etape 2: Installation de Create React App**
- **npm install -g create-react-app**

```
C:\Users\user>npm install -g create-react-app
npm warn deprecated uid-number@0.0.6: This package is no longer supported.
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated fstream-ignore@1.0.5: This package is no longer supported.
npm warn deprecated rimraf@2.7.1: Rimraf versions prior to v4 are no longer supported
npm warn deprecated fstream@1.0.12: This package is no longer supported.
npm warn deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.

added 64 packages in 6s

4 packages are looking for funding
  run 'npm fund' for details

C:\Users\user>
```



Installation de React

11

- **Etape 3: Création d'un nouveau projet React**
- **create-react-app my-app**

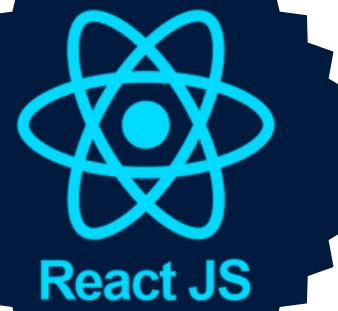
```
C:\Users\user>create-react-app mon_premier_projet_react
create-react-app is deprecated.
```

You can find a list of up-to-date React frameworks on react.dev
For more info see:<https://react.dev/link/cra>

This error message will only be shown once per install.

```
Creating a new React app in C:\Users\user\mon_premier_projet_react.
```

```
Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...
```



Installation de React

12

• Etape 4: Démarrage du serveur de développement

```
C:\Users\user>cd mon_premier_projet_react
C:\Users\user\mon_premier_projet_react>npm start
> mon_premier_projet_react@0.1.0 start
> react-scripts start

(node:11148) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:11148) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

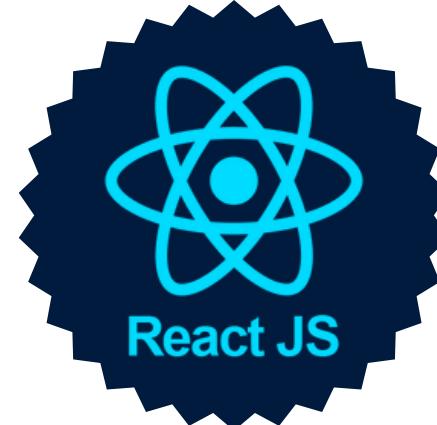
You can now view mon_premier_projet_react in the browser.

Local:          http://localhost:3000
On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
|
```

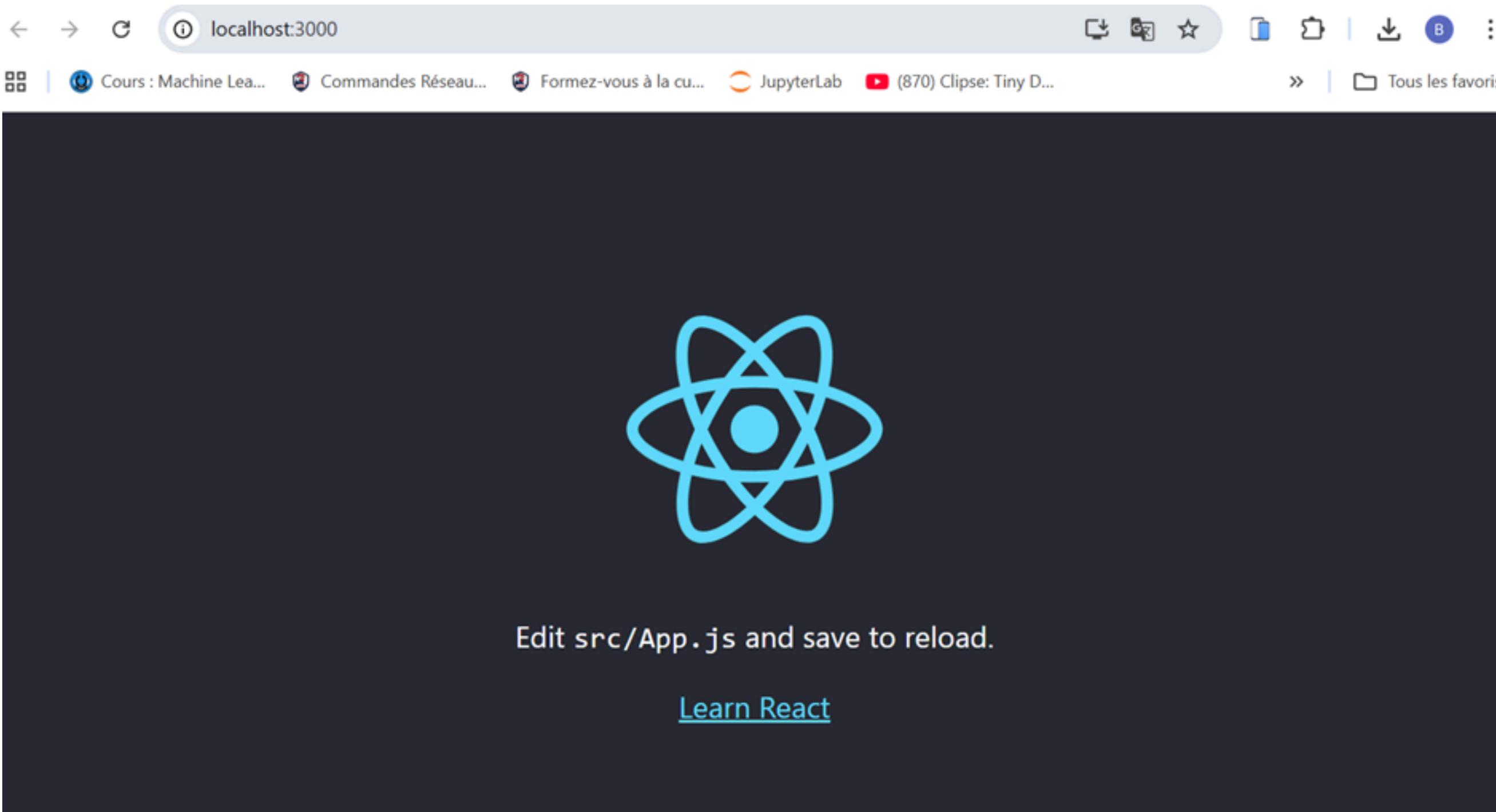
- **cd my-app**
- **npm start**

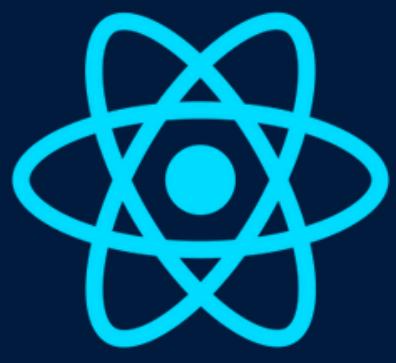


Installation de React

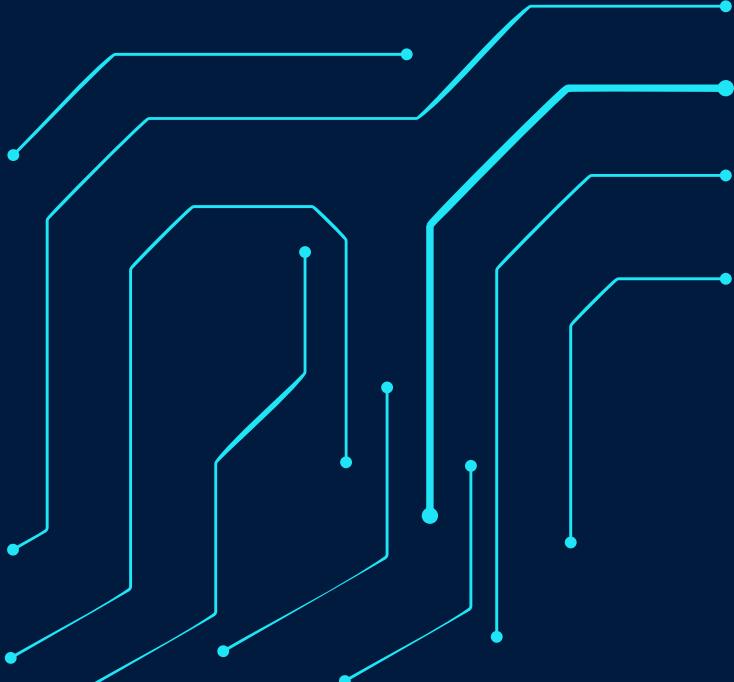
13

- **Etape 4: Démarrage du serveur de développement**



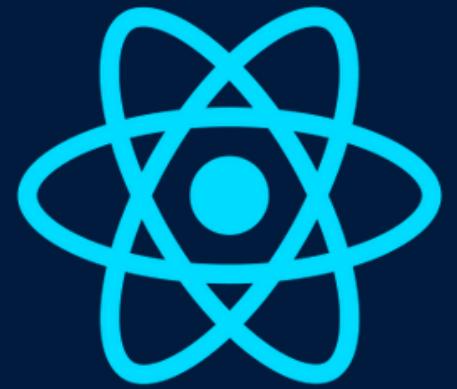


React JS



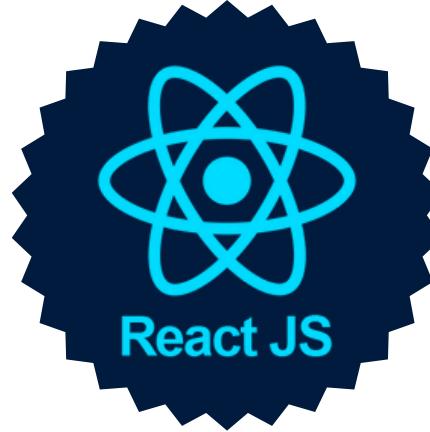
CONCEPTS





React JS

JSX (éléments, attributs, enfants, syntaxes à respecter, ...)



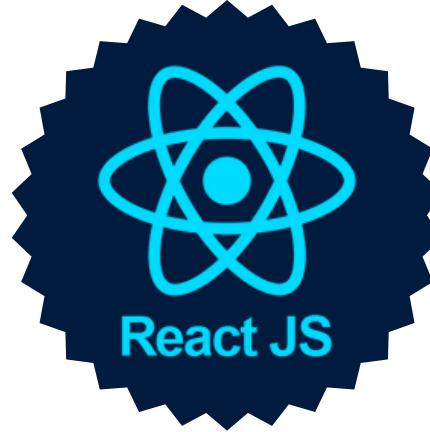
JSX: ELEMENT

16

- Le JSX (JavaScript XML) est une extension de syntaxe proposée par React.

Exemple:

```
const titre = <h1>Bonjour</h1>;
```



JSX: ELEMENT

17

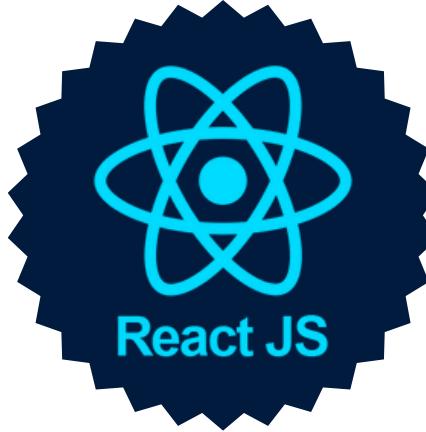
Caractéristiques:

Représentation du DOM virtuel :

- **Un élément JSX ressemble à une balise HTML, mais ce n'est pas vraiment du HTML.**

Exemple:

```
const titre = React.createElement("h1", null, "Bonjour");
```



JSX: ELEMENT

18

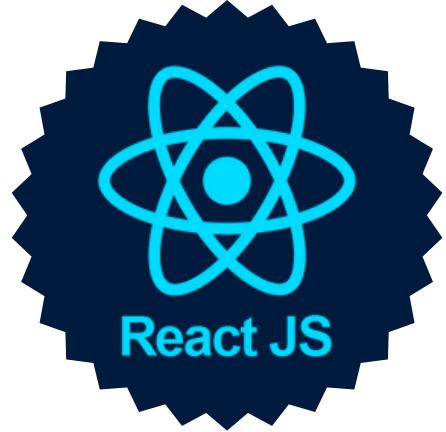
Caractéristiques:

Immuabilité:

- **Les éléments JSX sont immuables, c'est-à-dire qu'ils ne peuvent pas être modifiés une fois créés.**
- **Si l'application a besoin d'afficher une nouvelle information ou de modifier l'interface, React ne change pas l'élément existant : il crée un nouvel élément JSX**

Exemple:

```
const element = <h1>Bonjour</h1>;  
// On ne peut pas modifier "Bonjour" directement  
// Il faut recréer un nouvel élément  
const newElement = <h1>Bonsoir</h1>;
```



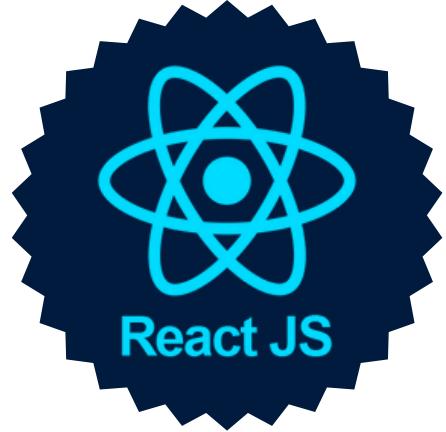
TYPES ELEMENTS

19

a) Éléments simples

- Un élément JSX simple est une balise unique écrite en JSX

```
const titre = <h1>Mon titre</h1>;  
const bouton = <button>Cliquez ici</button>;
```



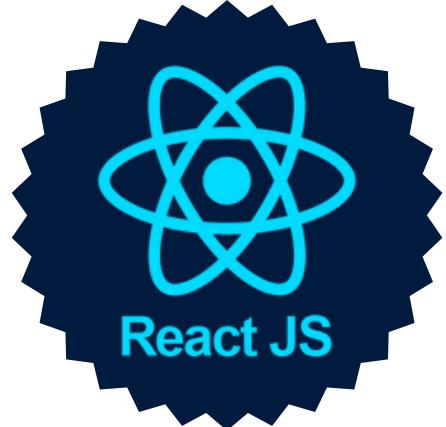
TYPES ELEMENTS

20

b) Éléments avec attributs

- **Un élément JSX avec attribut est une balise à laquelle on ajoute une propriété**

```
<label htmlFor="nom">Nom :</label>
<input id="nom" type="text" className="champ" />
```



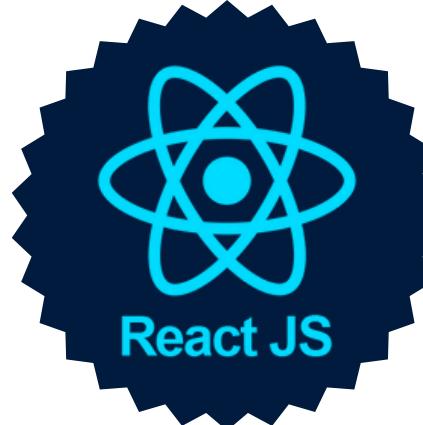
TYPES ELEMENTS (suite)

21

c) Éléments imbriques (enfants)

- **Un élément peut contenir d'autres éléments ou du texte.**

```
const card = (
  <div className="card">
    <h2>Titre</h2>
    <p>Ceci est un paragraphe enfant.</p>
  </div>
);
```



TYPES ELEMENTS (suite)

22

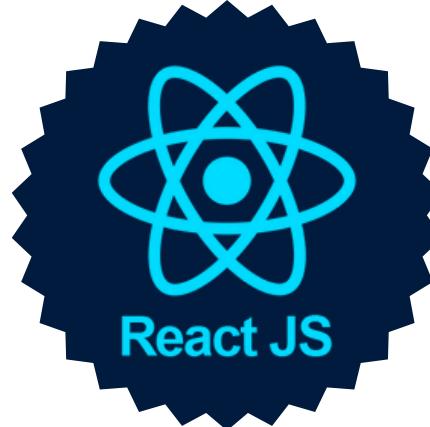
d) Éléments dynamiques avec expressions JavaScript

- On peut insérer du JS entre {}.
- mettre des calculs ou fonctions :

```
const utilisateur = "Omar";
const titre = <h1>Bienvenue {utilisateur}</h1>;
```

```
const nombre = 5;
const element = <p>2 + 3 = {2 + 3}</p>;
const heure = <p>Il est {new Date().toLocaleTimeString()}</p>;
```

```
function App() {
  const getHeure = () => new Date().toLocaleTimeString();
  return <p>Il est {getHeure()}</p>;
}
```



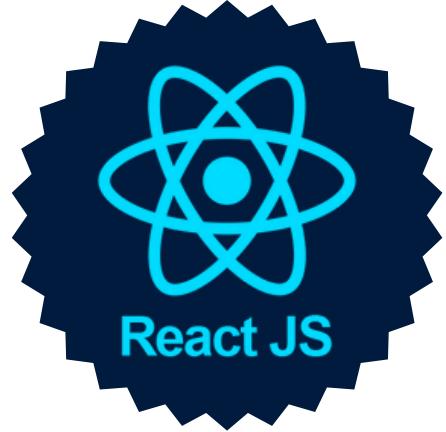
TYPES ELEMENTS (suite)

23

e) Éléments conditionnels

- On peut afficher un élément en fonction d'une condition :

```
const estConnecte = true;  
const message = estConnecte ? <h1>Bienvenue !</h1> : <h1>Veuillez vous connecter</h1>;
```



Règles de syntaxe JSX

24

- Chaque élément doit être fermé :

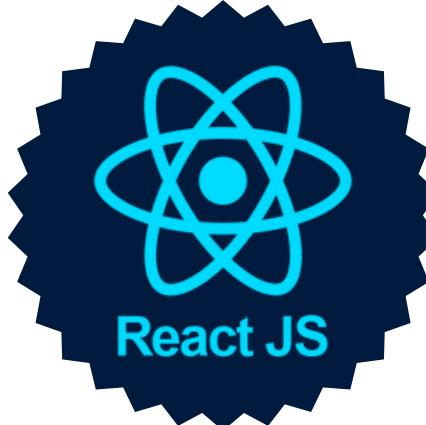
```
    // correct
<br />                      // correct
```

- Toujours retourner un seul parent :

```
return (
  <div>
    <h1>Titre</h1>
    <p>Texte</p>
  </div>
);
```

- Pour éviter une <div> inutile, on utilise les Fragments :

```
return (
  <>
    <h1>Titre</h1>
    <p>Texte</p>
  </>
);
```



ATTRIBUT JSX

25

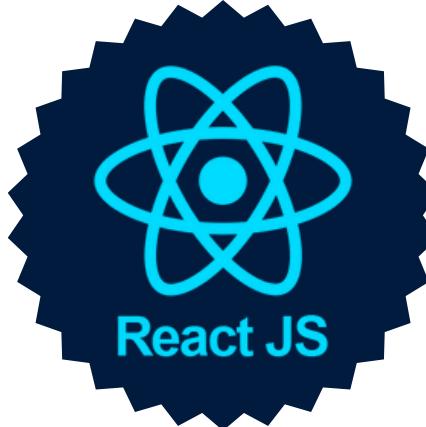
Expression comme attribut

- Utiliser des expressions JavaScript comme valeurs d'attribut, utile pour les attributs dynamiques.
- Si vous utilisez des guillemets, JSX la traitera comme une chaîne littérale et non comme une expression JavaScript.

Exemple

Utiliser des expressions JavaScript comme valeurs d'attribut :

```
function Car() {
  const x = "myclass";
  return (
    <h1 className={x}>Hello World</h1>
  );
}
```



ATTRIBUT JSX

26

- **Attributs d'événement camelCase**
- En JSX, les attributs d'événements (comme onclick, onchange en HTML) doivent être écrits en camelCase.

Exemple :

onsubmit=

onchange=

onclick=

onSubmit

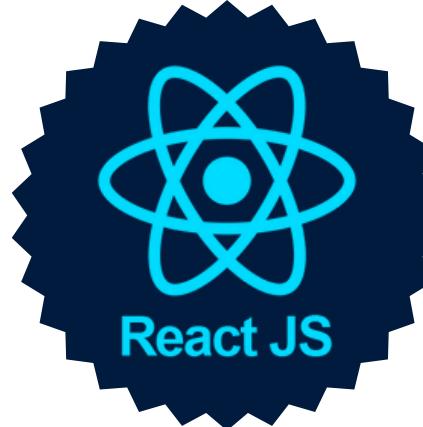
onChange

onClick

Exemple

Utilisez camelCase pour les attributs d'événement :

```
function Car() {  
  const myfunc = () => {  
    alert('Hello World');  
  };  
  return (  
    <button onClick={myfunc}>Click me</button>  
  );  
}
```



ATTRIBUTS SPECIAUX

- En JSX, en plus des attributs HTML classiques (id, src, alt, ...), il existe des attributs spéciaux que React utilise pour des raisons techniques.

Eviter les conflits avec JS

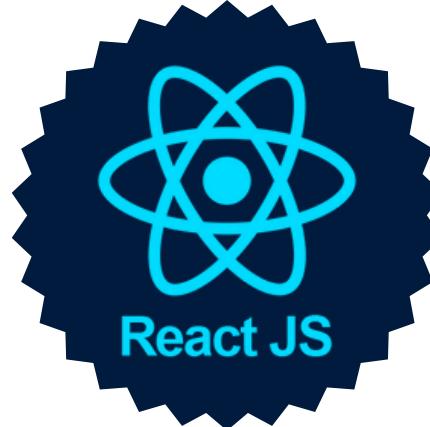
Exemple :

for=htmlFor
class=className

Exemple

Utiliser l'attribut `className` au lieu de `class` dans JSX :

```
function Car() {  
  return (  
    <h1 className="myclass">Hello World</h1>  
  );  
}
```



ATTRIBUT BOOLEEN

28

- Si vous ne transmettez aucune valeur à un attribut, JSX le considère comme **vrai**.

Exemple

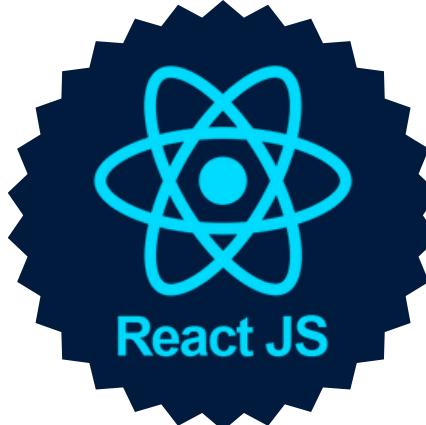
Booléen vrai en JSX, cela rendra le bouton désactivé :

```
<button onClick={myfunc} disabled>Click me</button>
```

Exemple

Également vrai en JSX, cela désactivera également le bouton :

```
<button onClick={myfunc} disabled={true}>Click me</button>
```



ATTRIBUT BOOLEEN

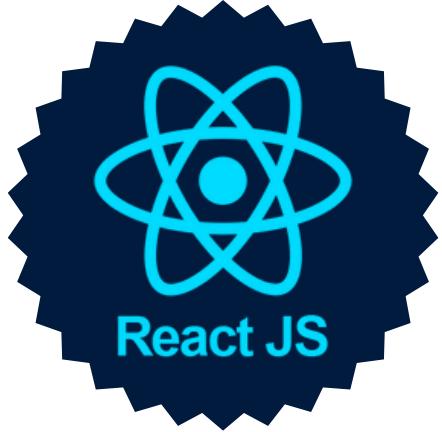
29

- Pour transmettre la valeur fausse, vous devez le spécifier sous forme d'expression.

Exemple

Faux en JSX, cela ne désactivera PAS le bouton :

```
<button onClick={myfunc} disabled={false}>Click me</button>
```



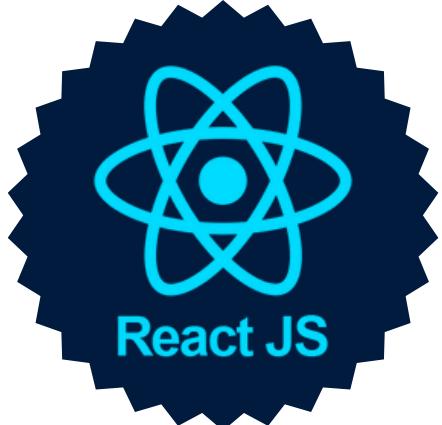
ATTRIBUT STYLE

30

- JSX est un peu spécial par rapport au HTML classique, car il utilise un objet JavaScript au lieu d'une chaîne de caractères.

En HTML:

```
<div style="color: red; font-size: 16px;">Texte</div>
```



ATTRIBUT STYLE

31

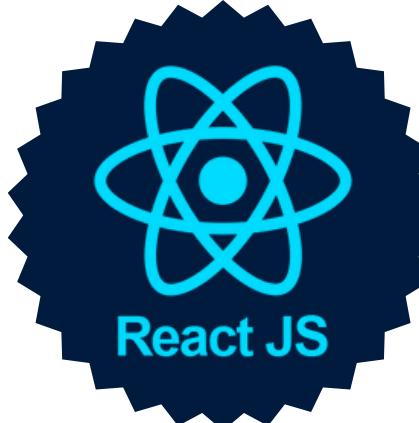
Les styles sont stockés dans un objet.

Les propriétés de style sont écrites en **camelCase** .

exemple **fontSize**, au lieu de **font-size**

Utilisez l' **style** attribut :

```
function Car() {  
  const mystyles = {  
    color: "red",  
    fontSize: "20px",  
    backgroundColor: "lightyellow",  
  };  
  
  return (  
    <>  
      <h1 style={mystyles}>My car</h1>  
    </>  
  );  
}
```



COMPOSANT

32

- Les composants sont comme des fonctions qui renvoient des éléments HTML.
- Un composant React est un morceau de code réutilisable qui représente une partie de l'interface utilisateur

Exemple

Créez un composant de fonction appelé **Car**

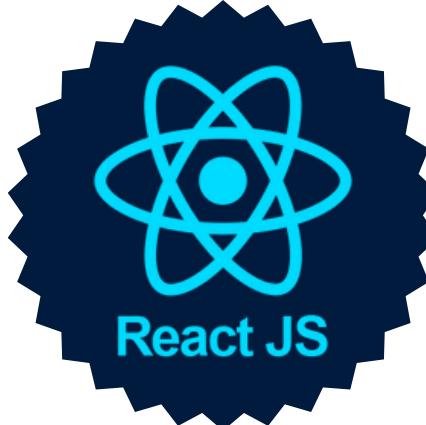
```
function Car() {  
  return (  
    <h2>Hi, I am a Car!</h2>  
  );  
}
```



Exemple

Afficher le **Car** composant dans l'élément « racine » :

```
createRoot(document.getElementById('root')).render(  
  <Car />  
)
```



COMPOSANT

33

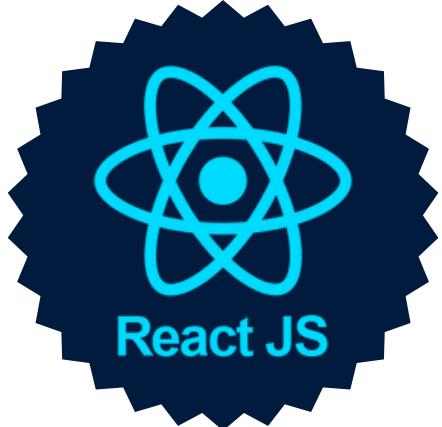
Les **props** sont des paramètres qu'on passe à un composant React pour le rendre dynamique et réutilisable

- Composant sans props

```
jsx  
  
function Bonjour() {  
  return <h1>Bonjour Omar</h1>;  
}
```

- Composant avec props

```
jsx  
  
function Bonjour(props) {  
  return <h1>Bonjour {props.nom}</h1>;  
}  
  
// Utilisation  
<Bonjour nom="Omar" />  
<Bonjour nom="Appoline" />
```



COMPOSANT

34

• COMPOSANT DANS LES COMPOSANTS

En React, un composant peut contenir ou utiliser d'autres composants à l'intérieur de son rendu.

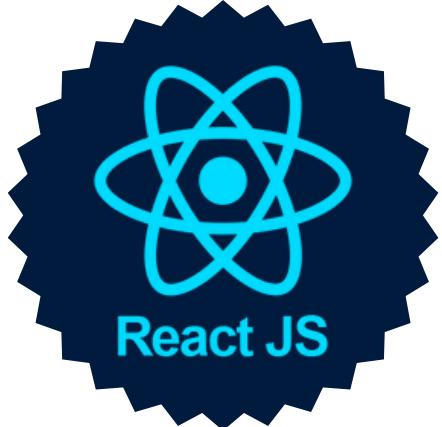
Utilisez le `Car` composant à l'intérieur du `Garage` composant :

```
function Car() {  
  return (  
    <h2>I am a Car!</h2>  
  );  
  
}  
  
function Garage() {  
  return (  
    <>  
      <h1>Who lives in my Garage?</h1>  
      <Car />  
    </>  
  );  
  
}  
  
createRoot(document.getElementById('root')).render(  
  <Garage />  
);
```

composant parent

composant enfant

#GroupeN°4



COMPOSANT

35

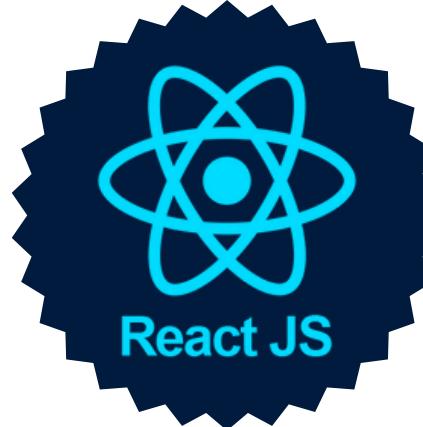
- **RENDU DE COMPOSANTS DEUX FOIS (ou plusieurs fois)**

Appeler un même composant plusieurs fois dans ton rendu, React va le créer et l'afficher autant de fois que tu l'écris.

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

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car brand="Ford" />
      <Car brand="BMW" />
    </>
  );
}

createRoot(document.getElementById('root')).render(
  <Garage />
);
```



COMPOSANT

36

• COMPOSANT DANS LES FICHIERS

React consiste à réutiliser du code, et cela peut être une bonne idée de diviser vos composants dans des fichiers séparés.

Exemple

Ceci est le nouveau fichier, nous l'avons nommé `Vehicle.jsx`:

Véhicule.jsx

```
function Car() {
  return (
    <h2>Hi, I am a Car!</h2>
  );
}

export default Car;
```

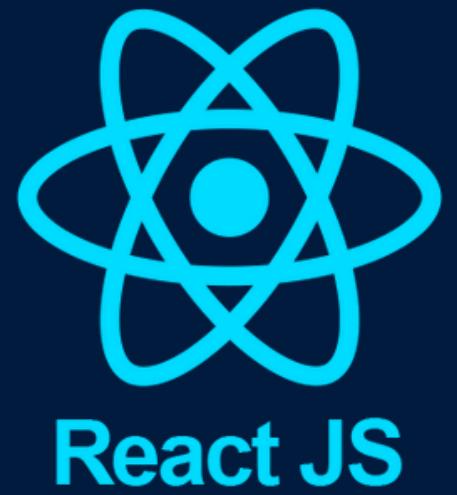
Exemple

Nous importons maintenant le `Vehicle.jsx` fichier dans l'application et nous pouvons utiliser

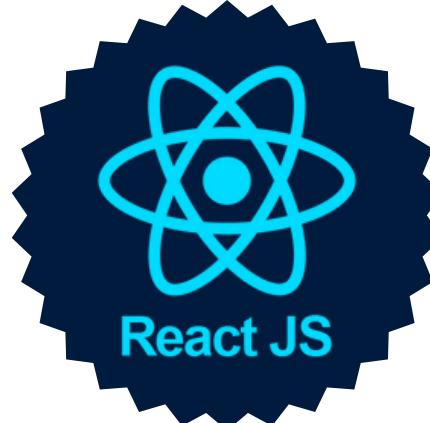
main.jsx

```
import { createRoot } from 'react-dom/client'
import Car from './Vehicle.jsx';

createRoot(document.getElementById('root')).render(
  <Car />
);
```



PROPS(propriété)



Qu'est ce qu'un Props ?

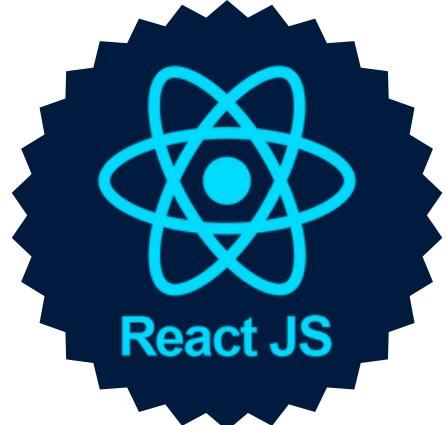
38

- les **Props** sont une sorte de paramètre qui règle un composant.
- Un prop en React, c'est un message ou une information que tu donnes à ton composant.

```
import { createRoot } from 'react-dom/client'

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

createRoot(document.getElementById('root')).render(
  <Car brand="Ford" model="Mustang" color="red" />
);
```



Exemple

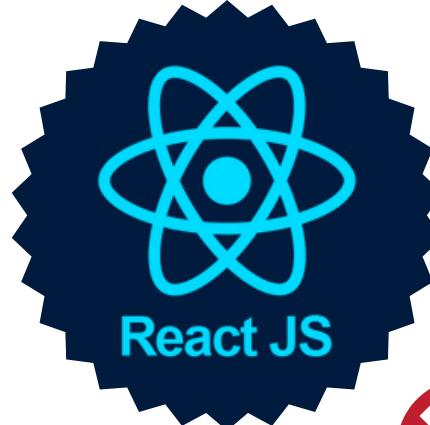
39

- Les props sont en lecture seule

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

function Garage() {
  return (
    <>
      <h1>Who lives in my Garage?</h1>
      <Car brand="Ford" />
      <Car brand="BMW" />
    </>
  );
}

createRoot(document.getElementById('root')).render(
  <Garage />
);
```



Pourquoi les Props ?

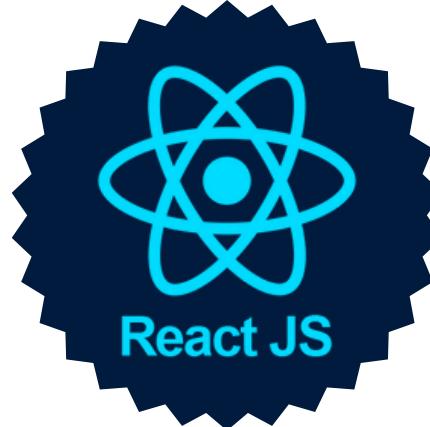
40

- ✗ Sans props, tous les composants afficheraient la même chose

```
function Smartphone() {  
  return <p>Samsung Galaxy S24 - 900€</p>;  
}  
  
export default function App() {  
  return (  
    <>  
      <Smartphone />  
      <Smartphone />  
      <Smartphone />  
    </>  
  );  
}
```

Les 3 affichent la même chose

#GroupeN°4



Pourquoi les Props ?

41



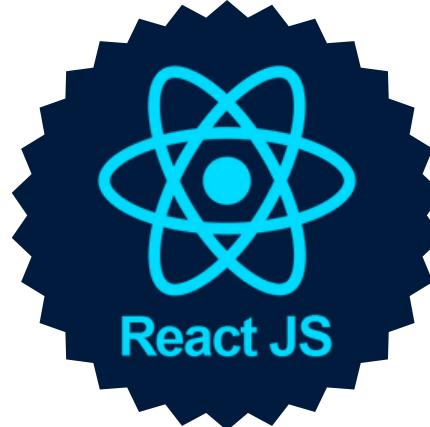
avec props, on peut lui passer des données pour qu'il s'adapte

```
function Smartphone({ marque, modele, prix }) {
  return <p>{marque} {modele} - {prix}</p>;
}

export default function App() {
  return (
    <>
      <Smartphone marque="Samsung" modele="Galaxy S24" prix={900} />
      <Smartphone marque="Apple" modele="iPhone 15" prix={1200} />
      <Smartphone marque="Xiaomi" modele="Mi 13" prix={700} />
    </>
  );
}
```

Résultat : Chaque smartphone a ses propres infos

#GroupeN°4



Pourquoi les Props ?

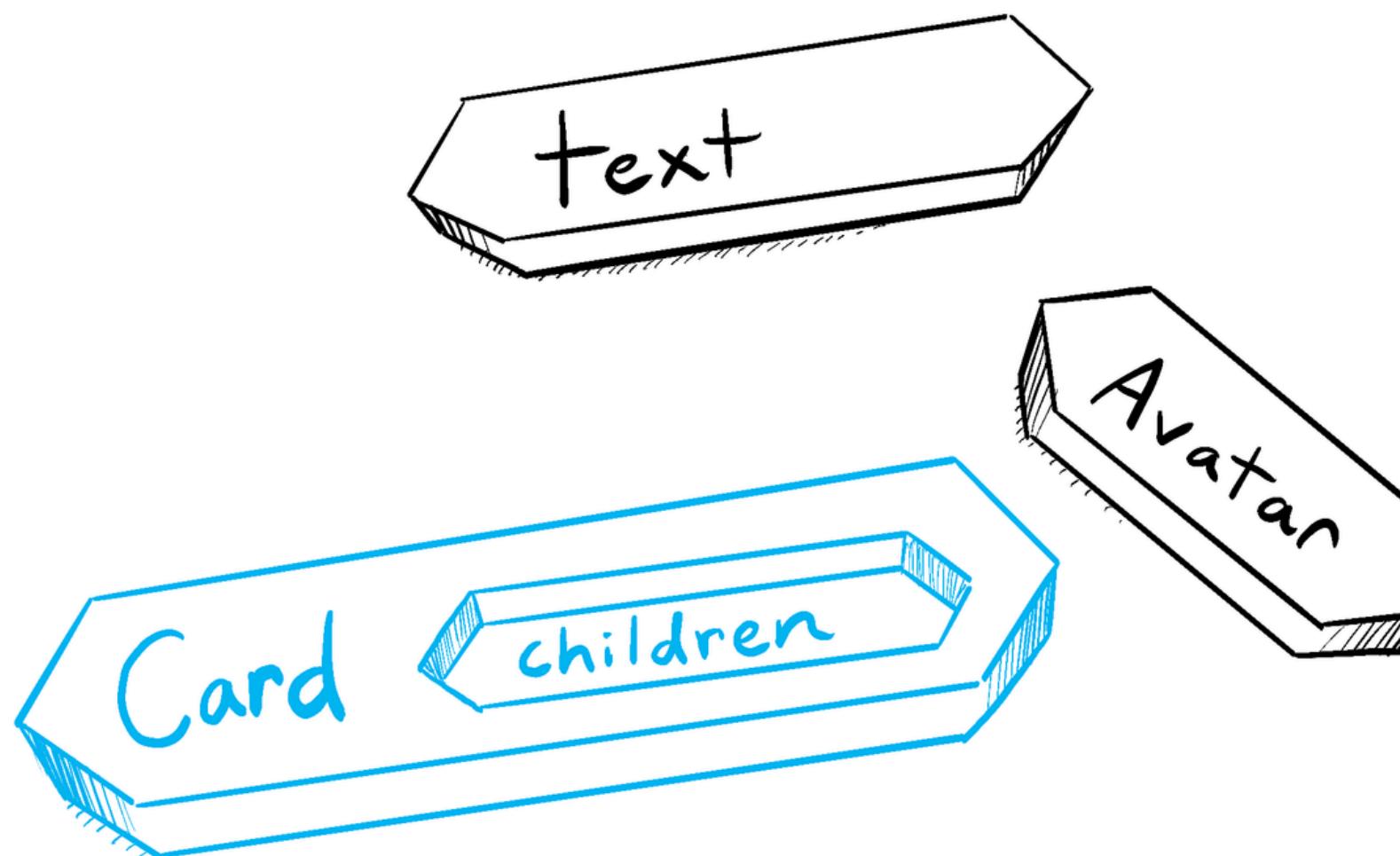
42

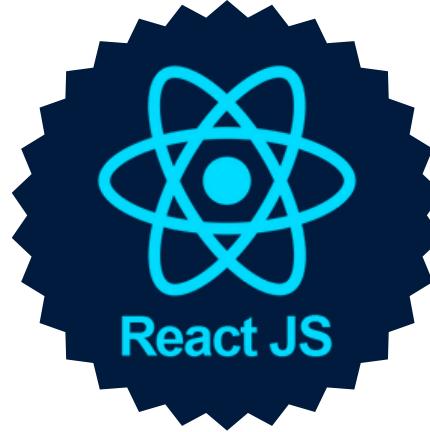


Tous les composants affichent la même chose



Rendre les composants réutilisables



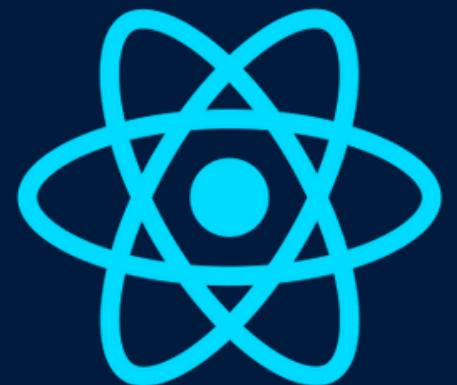


C'est quoi un Etat ?

43

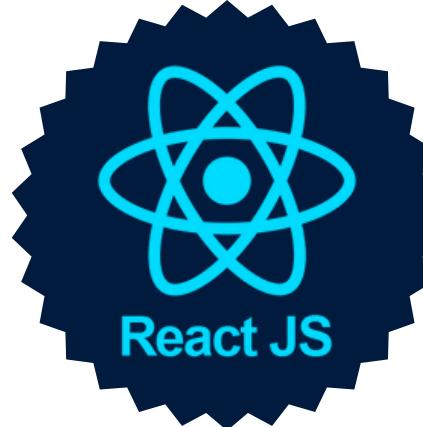
- En React, un état (**state**) est une valeur dynamique qui représente la situation actuelle d'un composant
- Contrairement aux props (valeurs fixes envoyées par un parent),
- le state peut changer avec le temps et déclenche automatiquement une mise à jour de l'interface (re-render).





React JS

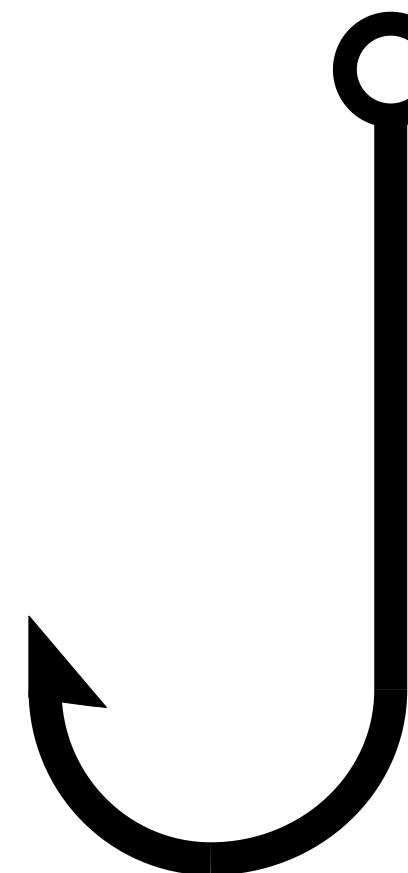
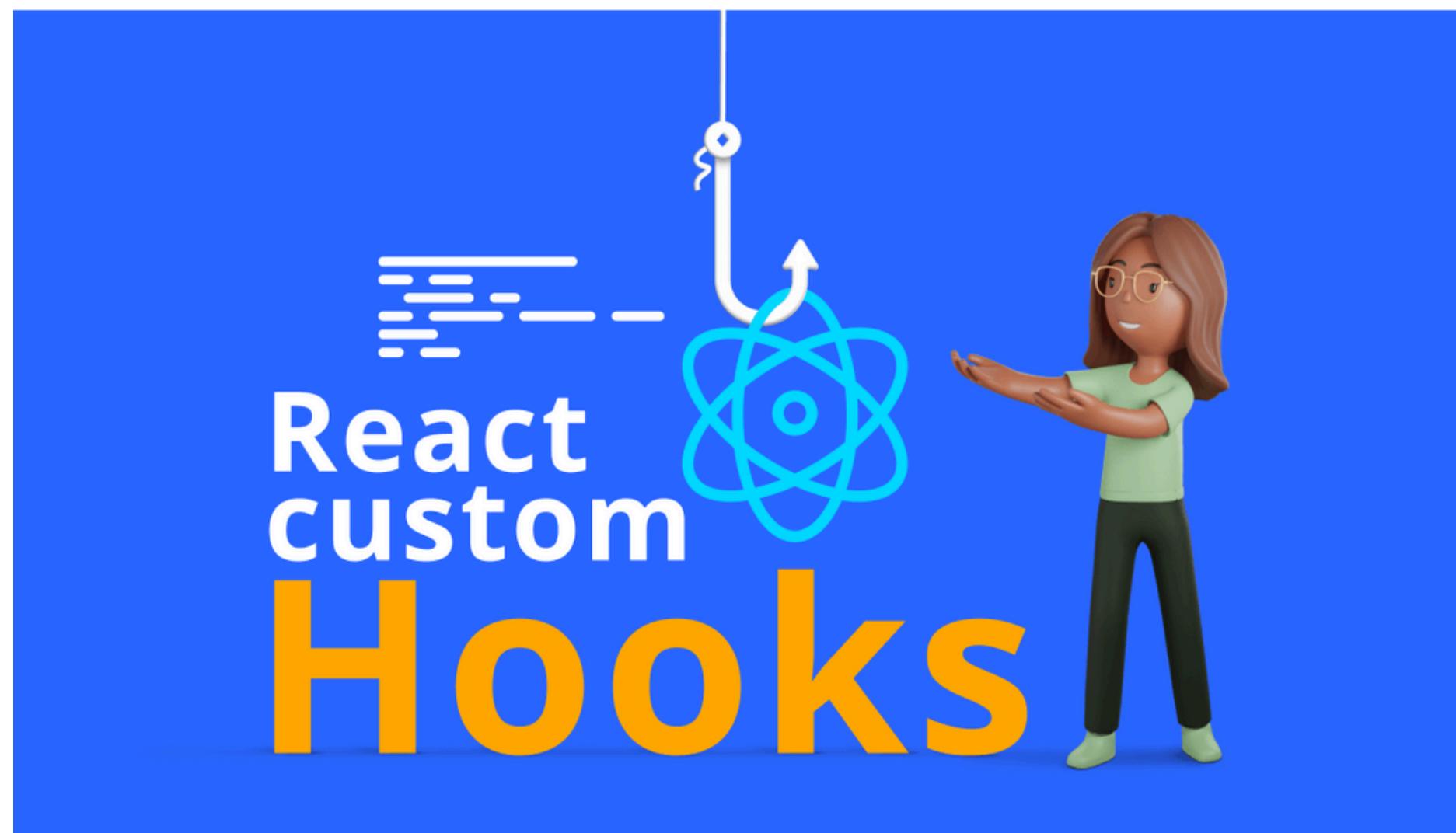
HOOKs/crochet



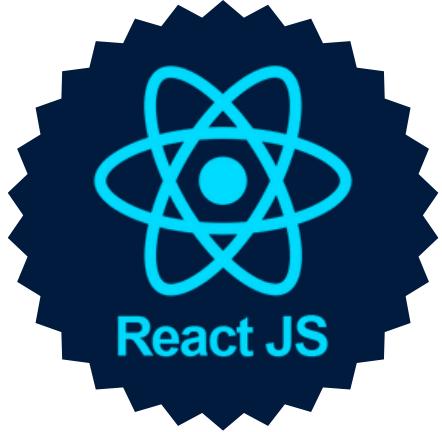
Qu'est ce qu'un HOOK ?

45

- **Un Hook en React, c'est juste un outil (une fonction spéciale) qui permet à ton composant :**
- **de se souvenir d'une information (exemple : un nombre, un texte, un état)**
- **ou de réagir à un changement (exemple : quand une donnée est mise à jour, quand la page charge, etc.).**



#GroupeN°4



Exemple • Avec le Hook useState

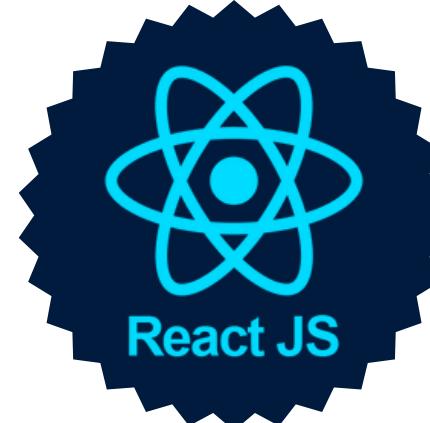
46

```
import { useState } from "react";

function GestionSmartphone() {
    // L'état (state) garde le nombre de smartphones
    const [nbSmartphones, setNbSmartphones] = useState(0);

    return (
        <div>
            <h2>Nombre de smartphones : {nbSmartphones}</h2>
            <button onClick={() => setNbSmartphones(nbSmartphones + 1)}>
                Ajouter un smartphone
            </button>
        </div>
    );
}
```

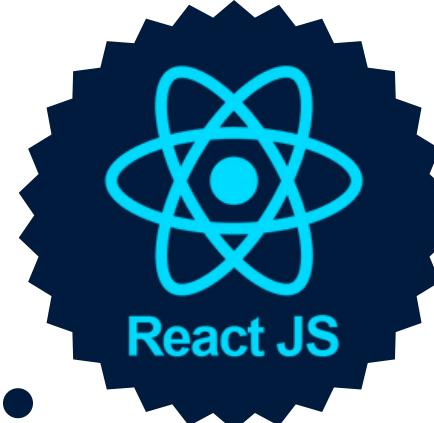
- **Donne une mémoire interne à ton composant (comme un petit carnet où il note une valeur qui peut changer).**
Exemple : compter le nombre de clics, garder un texte saisi.



Exemple

47

Hook	Utilité principale
useState	Gérer l'état local d'un composant (valeurs qui changent : compteur, formulaire, etc.).
useEffect	Effectuer des effets secondaires (exécuter du code après le rendu : appels API, timers , cycle de vie).
useContext	Consommer un contexte pour partager des données entre composants sans passer par les props .
useRef	Accéder directement à un élément du DOM ou stocker une valeur qui persiste sans provoquer de re-render .
useMemo	Mémoriser une valeur calculée pour éviter les recalculs inutiles et améliorer les performances.
useCallback	Mémoriser une fonction pour éviter qu'elle ne soit recréée à chaque rendu (optimisation).
useReducer	Gérer un état complexe avec une logique plus structurée (alternative avancée à useState).



Avant les HOOKs

48

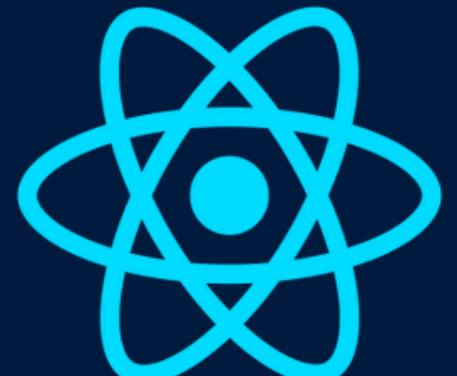
- Les composants fonctionnels étaient stateless (sans état).
- Ils ne pouvaient afficher que des données passées via les props.
- Pour gérer l'état ou les effets (comme déclencher une action quand le composant est monté), il faut utiliser les composants class.

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

class Compteur extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 }; // état
  }

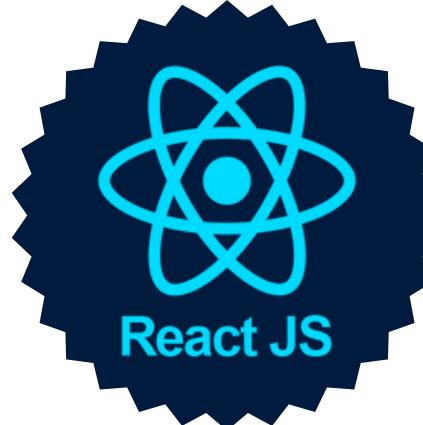
  componentDidMount() {
    console.log("Composant monté");
  }
}
```

this.state → état du composant
this.setState() → met à jour l'état
componentDidMount() → équivalent du montage (useEffect avec [])



React JS

Afficher une liste de composants



Afficher une liste de composants

50

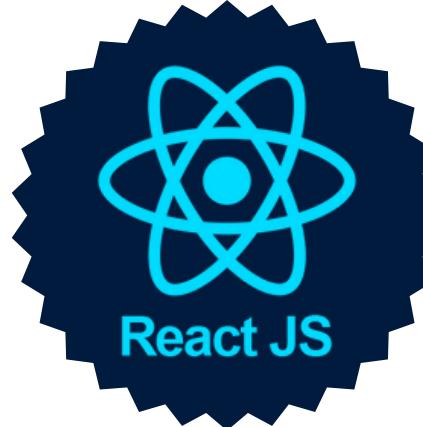
Répéter automatiquement un même composant plusieurs fois, à partir d'un tableau de données.

Exemple simple (explication sans code) :

- Tu as un tableau avec des noms : ["Ali", "Fatou", "Moussa"].
- Tu veux afficher chaque nom dans un composant <Utilisateur />.
- Au lieu d'écrire manuellement :

jsx

```
<Utilisateur name="Ali" />  
<Utilisateur name="Fatou" />  
<Utilisateur name="Moussa" />
```



Afficher une liste de composants

51

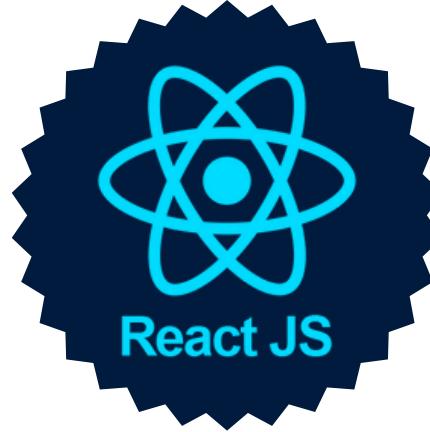
jsx

```
function ListePrenoms() {  
  const prenoms = ["Ali", "Fatou", "Moussa", "Awa"];  
  
  return (  
    <ul>  
      {prenoms.map((nom, index) => (  
        <li key={index}>{nom}</li>  
      ))}  
    </ul>  
  );  
  
  export default ListePrenoms;
```

const prenoms = [...] → tableau avec des prénoms.
.map() → parcourt chaque prénom et retourne un
****.
nom = l'élément actuel (ex : "Ali").
index = la position dans le tableau (0, 1, 2, ...), qu'on utilise comme key.



#GroupeN°4

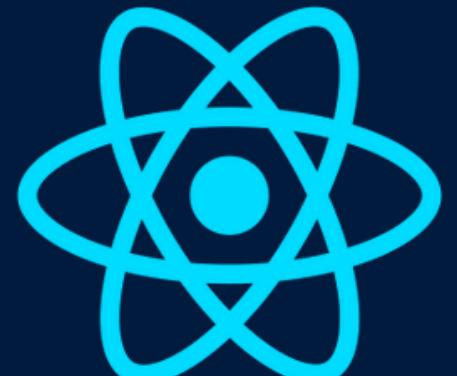


Resultat

52

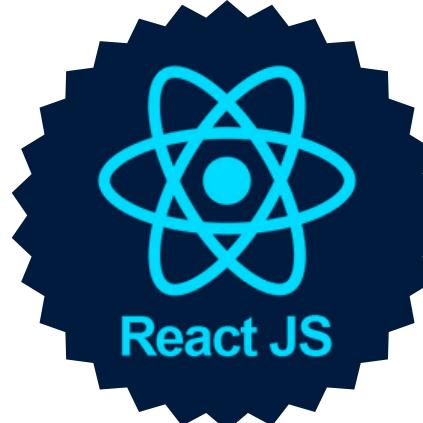
- Ali
- Fatou
- Moussa
- Awa

Ici chaque est un petit composant généré par **.map()**



React JS

Affichage conditionnelle

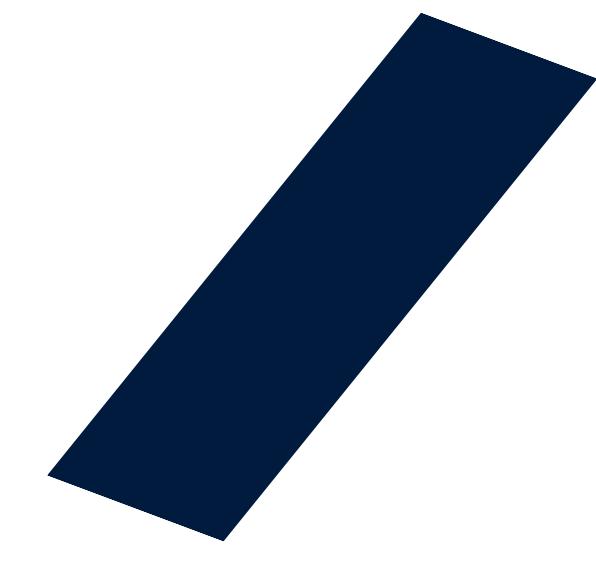


Affichage Conditionnelle

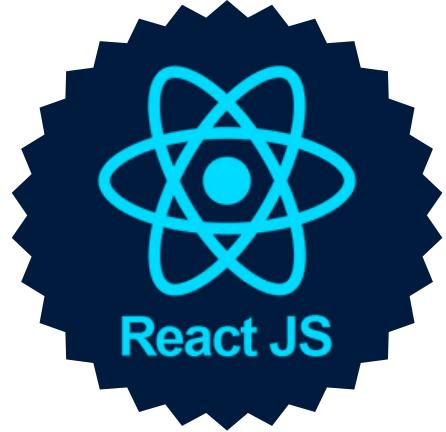
54

En React, on a souvent besoin d'afficher certains éléments uniquement si une condition est remplie. Cela fonctionne comme en JavaScript classique, mais appliqué dans le JSX.

**Si l'utilisateur
saisie ses
identifiants**



On distingue trois méthodes principales d'affichage conditionnelle :



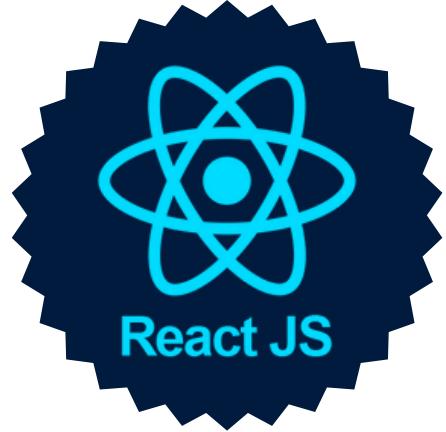
IF/ELSE (en dehors du JSX)

55

Idéal quand les branches retournent des blocs JSX très différents.

Exemple :

```
function MessageDeBienvenue({ estConnecte }) {
  if (estConnecte) {
    return <h1>Bienvenue !</h1>;
  } else {
    return <h1>Veuillez vous connecter.</h1>;
  }
}
```



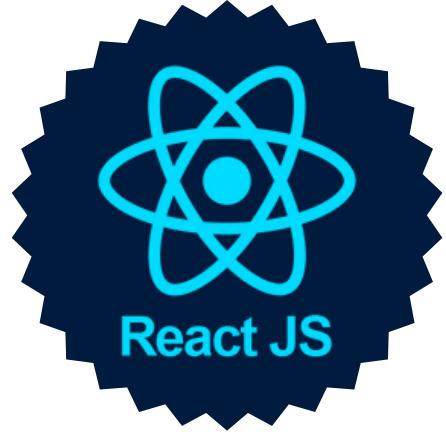
Opérateur ternaire ?:

56

Pratique pour un choix simple entre deux éléments.

Exemple :

```
function App() {
  const estConnecte = false;
  return (
    <div>
      {estConnecte ? <p>Bonjour utilisateur !</p> : <p>Veuillez vous connecter.</p>}
    </div>
  );
}
```



Opérateur logique &&

57

Parfait pour afficher quelque chose uniquement si la condition est vraie

Exemple :

```
function TableauDeBord({ hasNotification }) {  
  return (  
    <div>  
      <h1>Tableau de bord</h1>  
      {hasNotification && <p>Vous avez une nouvelle notification !</p>}  
    </div>  
  );  
}
```

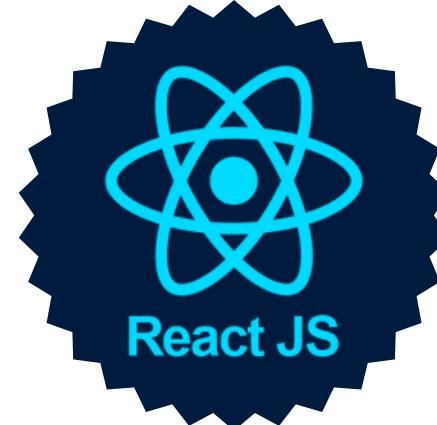
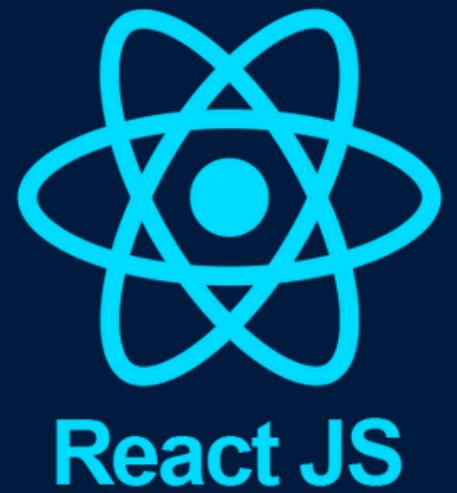


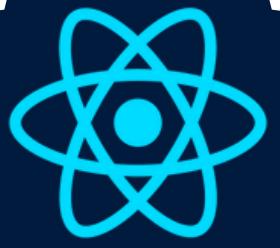
Tableau comparatif

58

Action	JS Vanilla	React
Écouter un clic	<code>addEventListener('click', fn)</code>	<code><button onClick={fn}></code>
Empêcher rechargement	<code>e.preventDefault()</code> (moderne) ou <code>return false</code> (obsolète)	<code>e.preventDefault();</code>
Lire la valeur d'input	<code>document.getElementById('input').value</code>	<code>useState + onChange</code>
Mise à jour de l'UI	Modification manuelle du DOM <code>:element.innerHTML = nouvelleValeur</code>	Mise à jour du state → re-render automatique



Événements



React JS

Exemple de gestionnaire d'événements

60

```
function ExempleComplet() {
  const [texte, setTexte] = useState('');

  const gererClic = () => alert('Clic !');

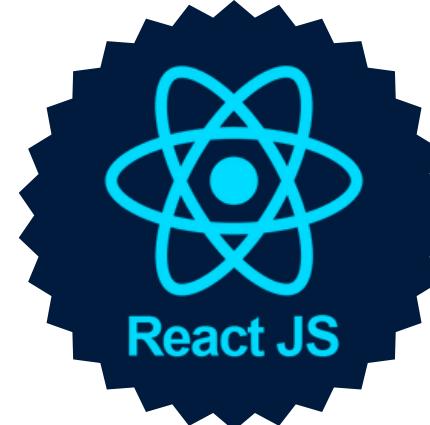
  const gererChangement = (event) => setTexte(event.target.value);

  const gererSoumission = (event) => {
    event.preventDefault(); // Empêche le rechargement
    console.log('Formulaire soumis avec :', texte);
  };

  return (
    <div>
      <button onClick={gererClic}>Clique-moi</button>

      <input type="text" value={texte} onChange={gererChangement} />

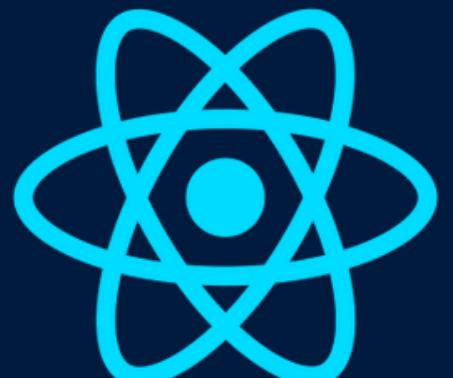
      <form onSubmit={gererSoumission}>
        <button type="submit">Envoyer</button>
      </form>
    </div>
  );
}
```



Points clés à retenir :

61

- En React, on décrit l'UI en fonction de l'état, pas besoin de manipuler directement le DOM.
- Affichage conditionnel : choisir dynamiquement quoi afficher (if/else, ternaire, &&).
- Gestion des événements : attacher directement les fonctions dans le JSX, mettre à jour le state → React gère la mise à jour de l'interface.
- L'avantage : logique et interface regroupées dans le même composant, plus clair et maintenable.

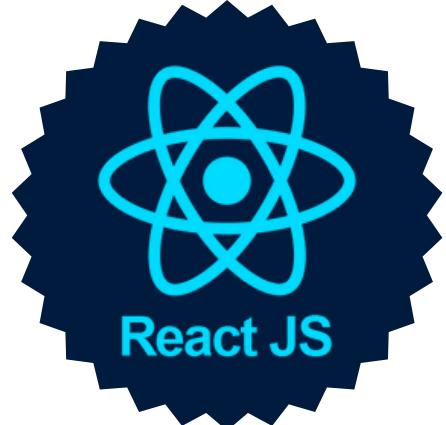


React JS

Formulaires

62

#GroupeN°4



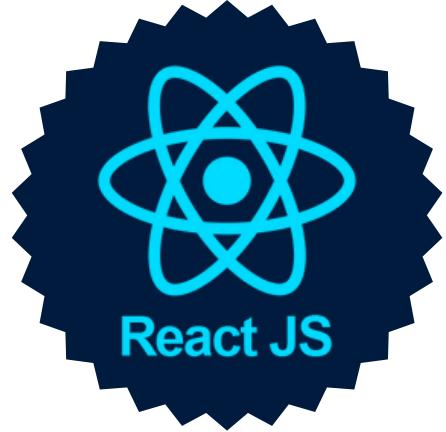
Definition Formulaire

63

les formulaires permettent de recueillir les données des utilisateurs, comme du texte, des nombres ou des sélections .

Ils fonctionnent comme des formulaires HTML, mais sont souvent contrôlés par l'état React, ce qui permet de suivre et de mettre à jour facilement les valeurs saisies.

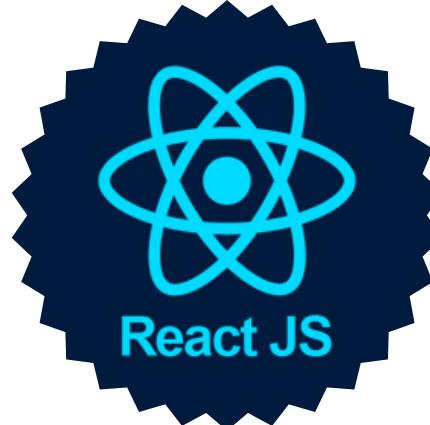
```
function App() {  
  return (  
    <form>  
      <label>  
        Nom :  
        <input type="text" name="name" />  
      </label>  
      <input type="submit" value="Envoyer" />  
    </form>  
  );  
}
```



Gestion des formulaires dans React

64

En React, les formulaires peuvent être gérés de deux façons : avec des composants contrôlés, où l'état du champ est géré par React, et avec des composants non contrôlés, où la valeur est directement gérée par le DOM.



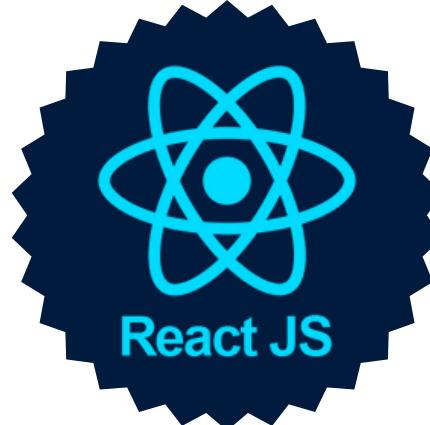
Composant Contrôlé

65

Un controlled input, c'est un input qui est géré par un state. La valeur affichée est totalement liée au state.

Si tu ne mets pas de readOnly, tu vas avoir un warning dans la console car tu n'as pas ajouté de onChange. L'utilisateur ne pourra pas modifier la valeur.

```
const Controlled = () => {
  return <input value="controlled" readOnly />;
};
```



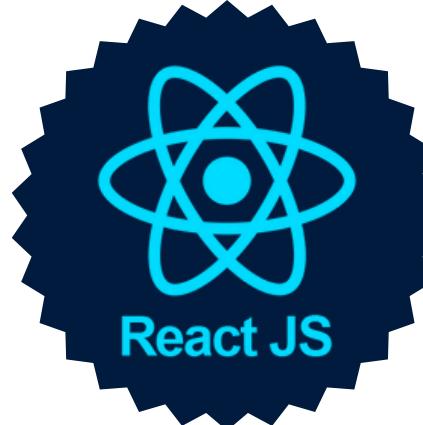
Composant Contrôlé :Exemple

66

Le controlled ne peut pas être modifié.

Le seul moyen de pouvoir le modifier est d'utiliser un useState React :

```
App.jsx X main.jsx
1 import {useState} from "react";
2
3 function App() {
4
5     const [firstname, setFirstname] = useState('John doe')
6
7     const handleChange = (e) => {
8         setFirstname(e.target.value)
9     }
10
11    return <form>
12        <input type="text" name="firstname" value={firstname}
13            onChange={handleChange}/>
14        {firstname}
15    </form>
16
17
18    export default App
19
```



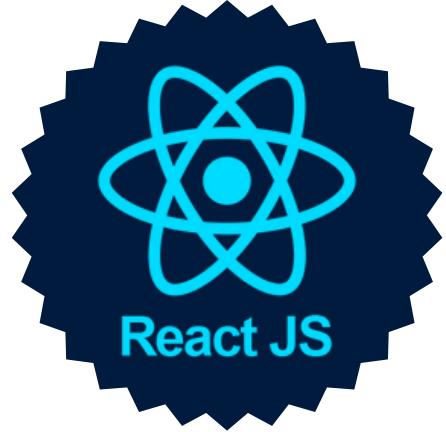
Composant Contrôlé:

Exemple avec textarea et checkbox

67

The screenshot shows a browser window with the URL `http://localhost:5173/?firstname=azeazeaze`. The developer tools are open, displaying the Elements and Console tabs. The Elements tab shows a form element with a text area and a checked checkbox. The Console tab shows three issues: 4 hidden, 3 levels, and 3 issues.

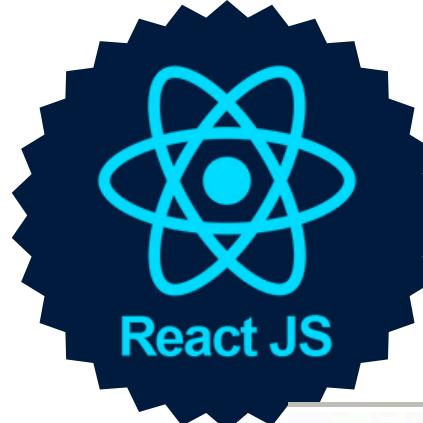
```
App.jsx X main.jsx
1 import {useState} from "react";
2
3 function App() {
4
5     const [value, setValue] = useState('')
6     const handleChange = (e) => {
7         setValue(e.target.value)
8     }
9
10    const [checked, setChecked] = useState(true)
11    const toggleCheck = () => {
12        setChecked(!checked)
13    }
14
15    return <form>
16        <textarea value={value} onChange={handleChange}>
17            <input type="checkbox" checked={checked} onChange={toggleCheck}> I
18            <button>Envoyer</button>
19        </form>
20    }
21
22
23 export default App
```



Composants Non Contrôlés

Dans ce cas, la valeur des champs est générée directement par le DOM, et React ne la suit pas en permanence.

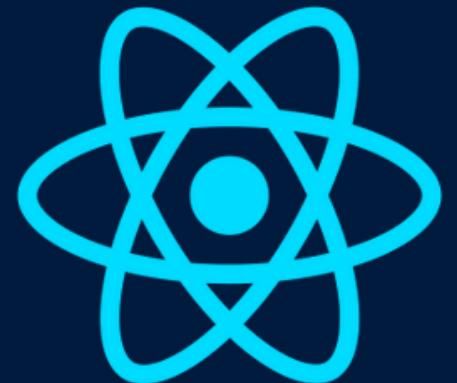
```
const Uncontrolled = () => {
  return <input defaultValue={'Uncontrolled'} />;
};
```



Composants Non Contrôlés : Exemple

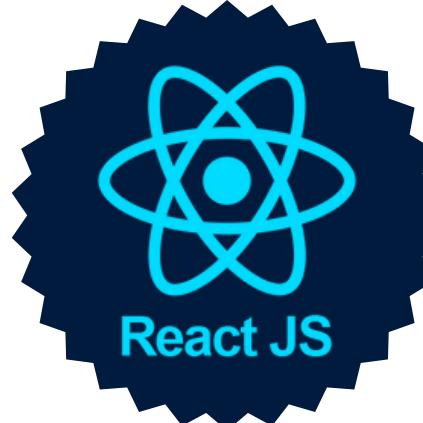
69

```
App.jsx X main.jsx
1 import {useState} from "react";
2
3 function App() {
4
5     const handleSubmit = (e) => {
6         e.preventDefault()
7         console.log(new FormData(e.target))
8     }
9
10    return <form onSubmit={handleSubmit}>
11        <input type="text" name="firstname" defaultValue="azeaze"/>
12        <button>Envoyer</button>
13    </form>
14}
15
16
17 export default App
18
```



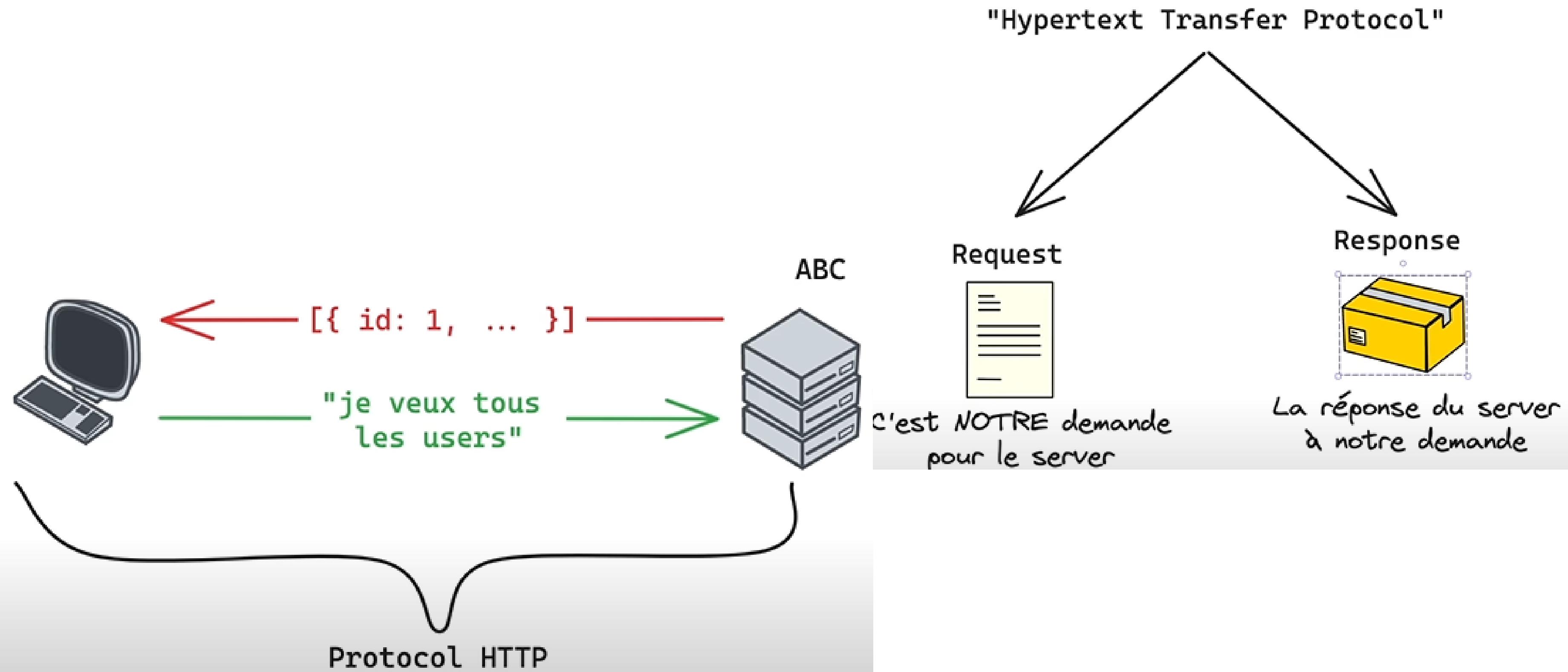
React JS

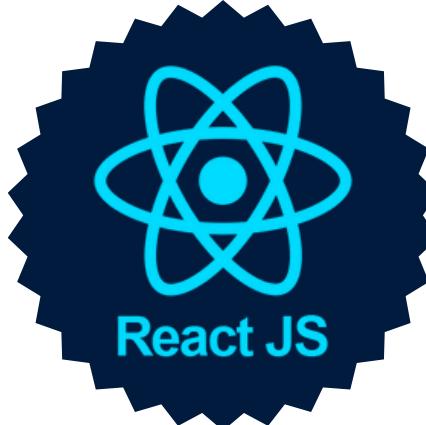
Communication avec un serveur HTTP



Protocole HTTP

71

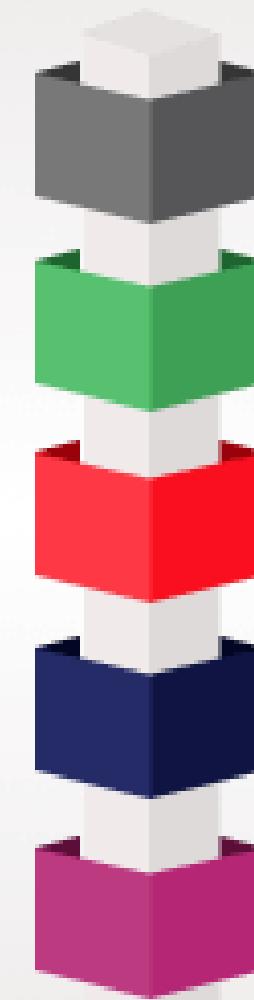




Protocole HTTP

72

HTTP Status Codes



1XX
INFORMATIONAL

2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

200 = Succès

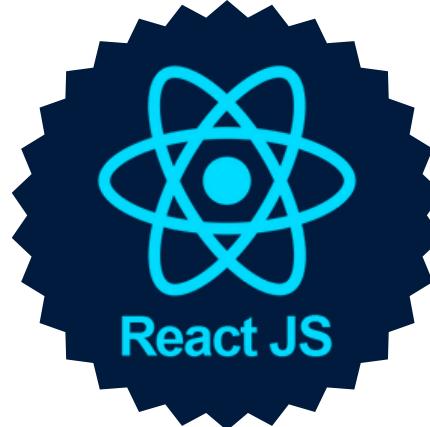
201 = Élément créé

400 = Erreur dans la request

401 = Pas autorisé

404 = Not found

500 = le server à crash

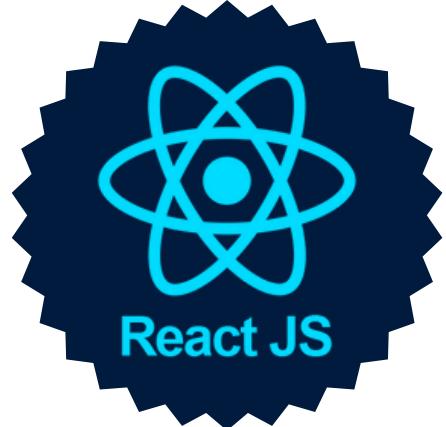


Intégration des requêtes HTTP dans une application React

73

L' API Fetch pour effectuer des requêtes HTTP et traiter les réponses.

```
await fetch(` ${API_URL}/${id}` , {  
  method: 'PUT',  
  headers: { 'Content-Type': 'application/json' },  
  body: JSON.stringify(data)  
});  
}
```



Gestion des requêtes HTTP avec fetch

74

Une Promesse (Promise): Utilisée avec fetch pour attendre la réponse du serveur sans bloquer l'application

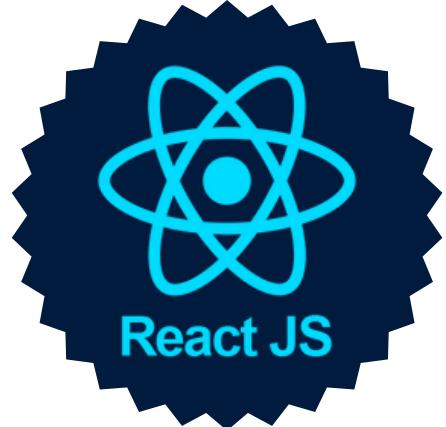
The screenshot shows a browser's developer tools console with the 'Console' tab selected. The command `const result = fetch("https://jsonplaceholder.typicode.com/posts")` is entered, followed by its return value `undefined`. Then, `result` is checked, revealing it is a `Promise` object. Finally, another `fetch` call is made, and its return value is shown as a pending promise.

```
> const result = fetch("https://jsonplaceholder.typicode.com/posts")
< undefined
> result
< ▶ Promise {<fulfilled>: Response}
> fetch("https://jsonplaceholder.typicode.com/posts")
< ▶ Promise {<pending>}
```

Représente une opération asynchrone

Peut être :

- ✓ Résolue (succès → on reçoit la donnée)
- ✗ Rejetée (échec → erreur)



Gestion des requêtes HTTP avec fetch :

GET

75

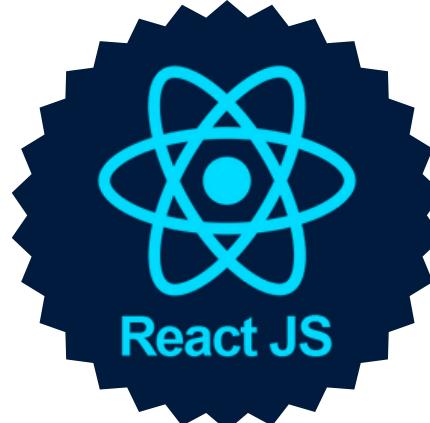
une requête GET pour récupérer des données depuis l'API sous format JSON.

```
> fetch("https://jsonplaceholder.typicode.com/posts")
    .then(res => {
      return res.json()
    })
    .then(json => console.log(json))
```

← ▶ Promise {<pending>}

[VM170:5](#)

```
(100) [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}],
[{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}], [{}]
```



Gestion des requêtes HTTP avec fetch :

GET

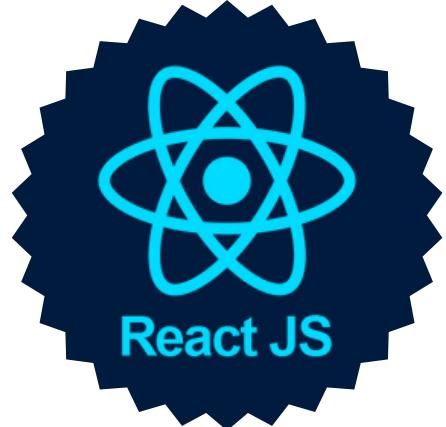
76

une requête GET pour récupérer des données depuis l'API en texte brut.

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(res => res.text()) // récupère la réponse en texte brut
  .then(text => console.log(text));

  ▶ Promise {<pending>}

[VM210:3
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\\nsuscipit recusandae consequuntur expedita et cum\\nreprehenderit molestiae ut ut quas totam\\nnostrum rerum est autem sunt rem eveniet architecto"
},
{
  "userId": 1,
  "id": 2,
  "title": "qui est esse",
  "body": "est rerum tempore vitae\\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\\nqui aperiam non debitis possimus qui neque nisi nulla"
},
```



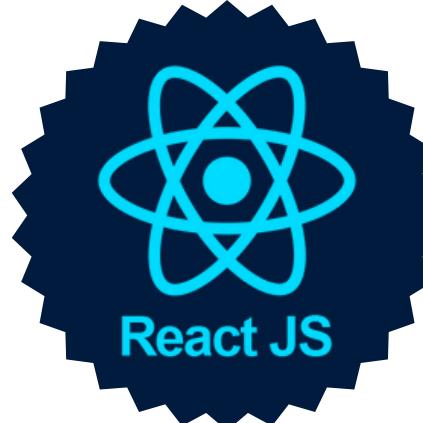
Gestion des requêtes HTTP avec fetch : POST

77

**Une requête POST pour
envoyer des données au
serveur sous forme de
JSON.**

```
> fetch("https://jsonplaceholder.typicode.com/posts", {  
    method: "POST", // type de requête  
    body: {  
        title: 'foo',  
        body: 'bar',  
        userId: 1,  
    }  
})
```

```
< ▾ Promise {<pending>} ⓘ  
  ▶ [[Prototype]]: Promise
```



Gestion des requêtes HTTP avec fetch : POST

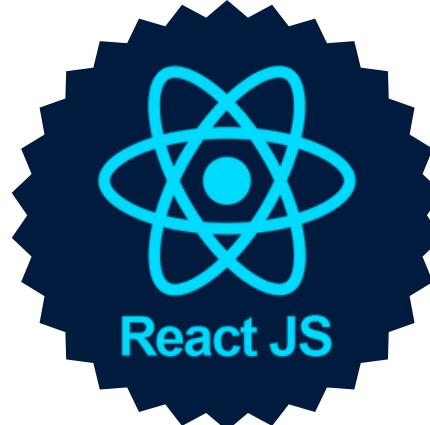
78

Si on envoie directement un objet JavaScript dans le body, le serveur reçoit [object Object] car les données ne sont pas converties en JSON lisible.

The screenshot shows the Network tab in the Chrome DevTools developer console. The timeline at the top indicates a 100 ms duration. Below it, a list of network requests is shown, with one entry expanded to show its details. The expanded entry is for a POST request named 'posts'. In the 'Payload' section, the value '[object Object]' is displayed, indicating that the raw JavaScript object was sent without being converted to JSON.

Name	Headers	Payload	Preview	Response	More
posts		[object Object]			>>
posts					
posts					

#GroupeN°4



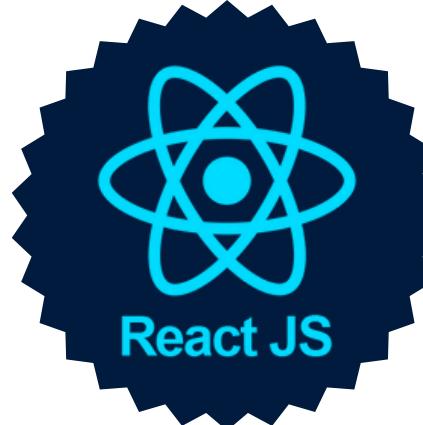
Gestion des requêtes HTTP avec fetch : POST

79

Pour rendre le contenu visible et exploitable, il faut utiliser JSON.stringify() avant l'envoi. »

```
> fetch("https://jsonplaceholder.typicode.com/posts", {  
  method: "POST", // type de requête  
  body:JSON.stringify( {  
    title: 'foo',  
    body: 'bar' ,  
    userId: 1 ,  
  })  
})
```

```
< ➔ Promise {<pending>}
```



Gestion des requêtes HTTP avec fetch : POST

80

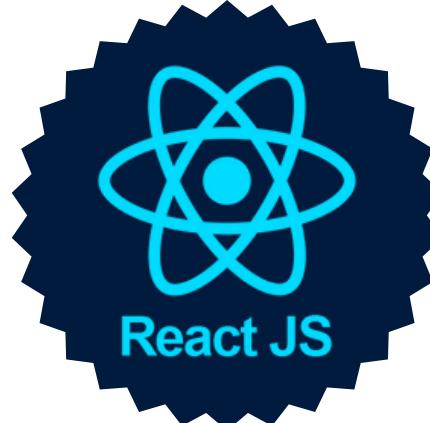
Résultat de l'envoie dans Payload:

Name	X	Headers	Payload	Preview	Response	>
posts						
posts						
posts						
posts						
posts						

Request Payload [View source](#)

```
{title: "foo", body: "bar", userId: 1}  
body: "bar"  
title: "foo"  
userId: 1
```

7 requests | 10.5 kB transferred



Gestion des requêtes HTTP avec fetch : POST

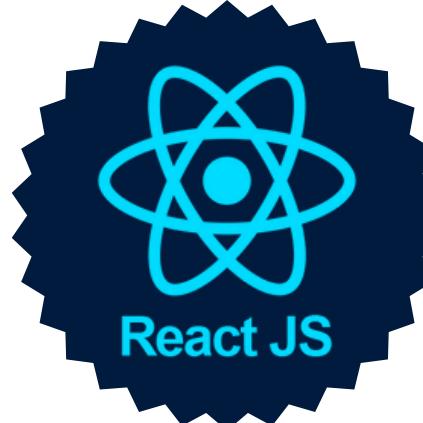
81

Une requête POST pour envoyer des données au serveur avec des en-têtes (headers) personnalisés sous forme de paires clé-valeur.

```
> fetch("https://jsonplaceholder.typicode.com/posts", {
    method: "POST", // type de requête
    body: JSON.stringify( {
        title: 'foo',
        body: 'bar' ,
        userId: 1 ,
    }),
    headers: {
        "key": "value",
        "Prenom": "Aminata",
    },
}).then(res => console.log(res)).catch
< f catch() { [native code] }

VM202:12
▶ Response {type: 'basic', url: 'https://jsonplaceholder.typicode.com/po
sts', redirected: false, status: 201, ok: true, ...}

>
```



Gestion des requêtes HTTP avec fetch : POST

82

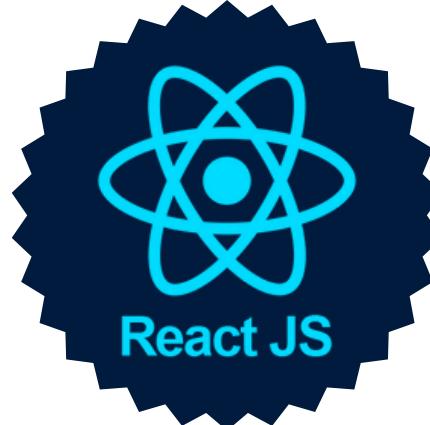
Informations retournées dans les headers :

ame	X	Headers	Payload	Preview	Response	>>
posts		Key		value		
posts		Origin		https://jsonplaceholder.typicode.com		
posts		Prenom		Aminata		
posts		Priority		u=1, i		
posts		Referer		https://jsonplaceholder.typicode.com/posts		
users		Sec-Ch-Ua		"Not;A=Brand";v="99", "Google Chrome";v="139", "Chromium";v="139"		
users		Sec-Ch-Ua-Mobile		?0		
posts		Sec-Ch-Ua-Platform		"Windows"		
posts		Sec-Fetch-Dest		empty		

13 requests

13.9 kB transferred

#GroupeN°4



Gestion des requêtes HTTP avec fetch : DELETE

83

Une requête DELETE pour supprimer des données sur le serveur.

```
> fetch("https://jsonplaceholder.typicode.com/posts", {  
    method: "DELETE", // type de requête  
}).then(res => res.json()).then(json => console.log(json))
```

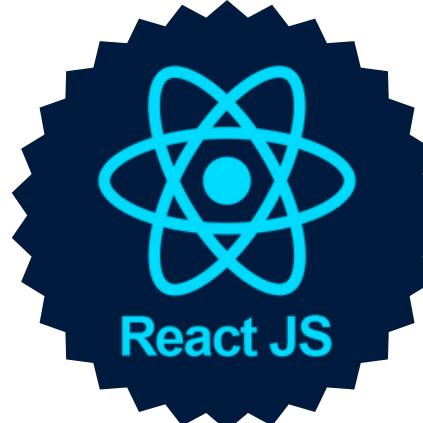
```
<  ▶ Promise {<pending>}
```

```
✖  ▶ DELETE https://jsonplaceholder.typicode.com/posts 404 (Not Found)
```

```
▶ {}
```

[VM182:3](#)

```
>
```



Gestion des requêtes HTTP avec fetch : DELETE

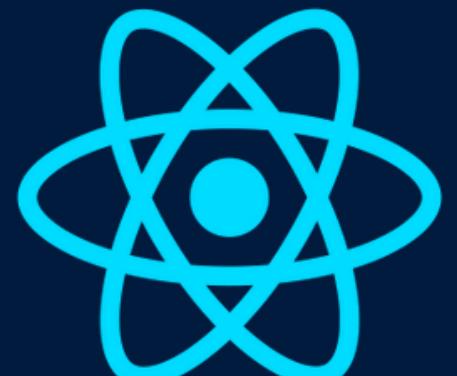
84

Résultat de la requête DELETE pour supprimer des données sur le serveur.

Name	X Headers	Preview	Response	Initiator	>>
posts	Request URL		https://jsonplaceholder.typicode.com/posts		
posts	Request Method		DELETE		
posts	Status Code		404 Not Found		
posts	Remote Address		104.21.80.1:443		
posts	Referrer Policy		strict-origin-when-cross-origin		
posts	▼ Response Headers				
posts	Access-Control-Allow-Credentials		true		
posts	Access-Control-Allow-Origin		https://jsonplaceholder.typicode.com		

8 requests

11.3 kB transferred

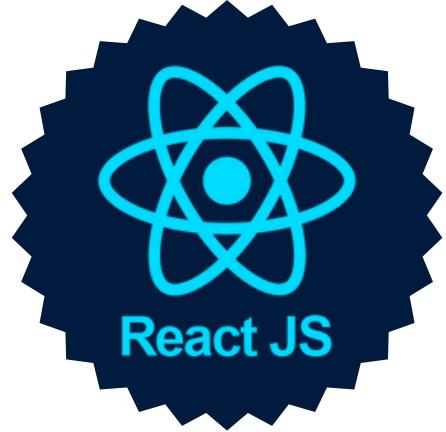


React JS

Routage

85

#GroupeN°4

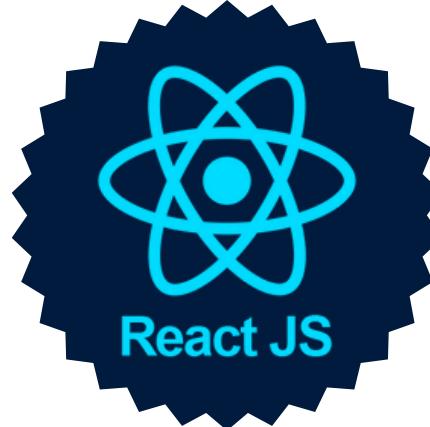


Routage :Définition

86

Le routage permet de gérer la navigation entre différentes pages ou vues d'une application React sans recharger toute la page.

React étant une Single Page Application (SPA), il n'y a qu'une seule page HTML. Le routage permet de simuler plusieurs "pages" en fonction de l'URL.



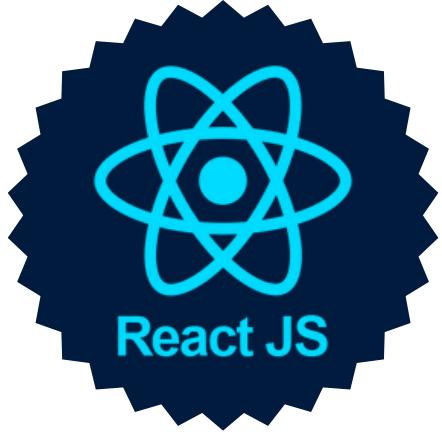
Qu'est-ce que React Router ?

87

React Router est la bibliothèque de routage standard pour les applications React. Elle permet :

- Offrir une expérience fluide : naviguer sans rechargement complet.
- Afficher des composants différents en fonction de l'URL.
- Gérer facilement la navigation (menu, liens, pages dynamiques).

Sans routeur, votre application React serait limitée à une seule page sans aucun moyen de naviguer entre différentes vues.

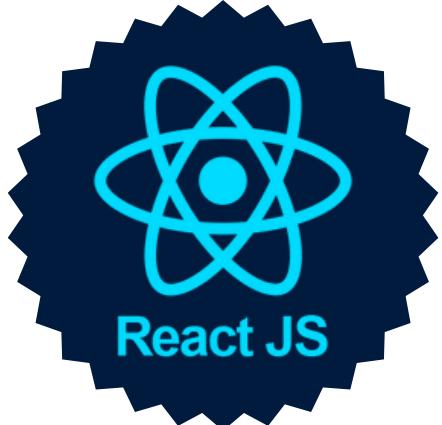


Syntaxe de base

88

Votre application doit être encapsulée avec le BrowserRouter composant pour activer le routage :

```
function App() {
  return (
    <BrowserRouter>
      {/* Your app content */}
    </BrowserRouter>
  );
}
```

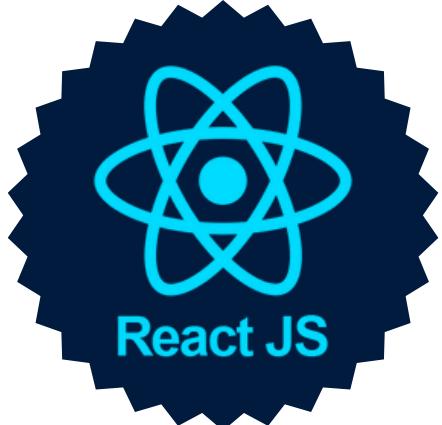


Exemple

89

- **BrowserRouter enveloppe votre application et active la fonctionnalité de routage**
- **Routes et Route définissez votre configuration de routage**

```
export default function App() {
  return (
    <BrowserRouter>
      <div className="flex flex-col min-h-screen">
        <Header/>
        <main className='flex-grow p-4 bg-gray-100'>
          <Routes>
            <Route path="/" element={<Classe />} />
            <Route path="/ajouter" element={<AjouterSmartphone />} />
            <Route path="/detail/:id" element={<DetaillerSmartphone />} />
            <Route path="/edit/:id" element={<EditerSmartphone />} />
          </Routes>
        </main>
      </div>
      <Footer />
    </BrowserRouter>
  );
}
```



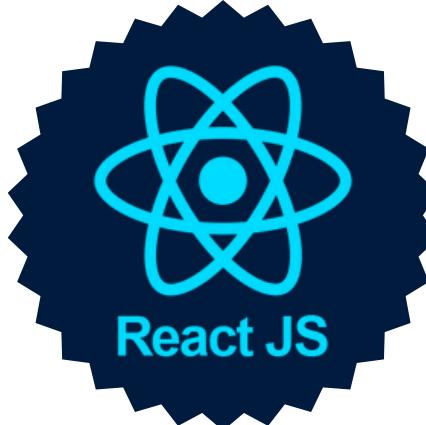
Exemple

90

<Link> est un composant de React Router qui permet de naviguer entre les routes de l'application sans recharger la page.

```
import { Link } from 'react-router-dom';

export default function Header() {
  return (
    <header className="text-white bg-gray-900 shadow-md">
      <div className="container flex items-center justify-between p-4 mx-auto">
        <h1 className="text-xl font-bold">Gestionnaire de smartphones</h1>
        <nav>
          <ul className="flex gap-4">
            <li><Link to="/" className="hover:text-blue-400">Accueil</Link></li>
            <li><Link to="/ajouter" className="hover:text-blue-400">Ajouter un Smartphone</Link></li>
          </ul>
        </nav>
      </div>
    </header>
  );
}
```



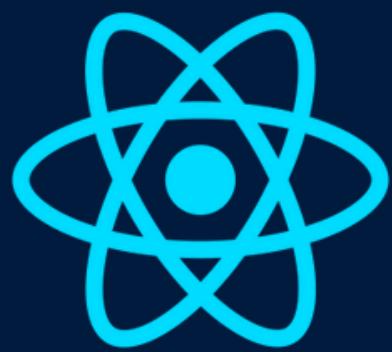
Exemple

91

Importer les bibliothèques dans App.js permet d'utiliser React Router pour définir et gérer les routes de l'application.

A screenshot of a code editor showing the contents of the App.jsx file. The file imports several components from 'react-router-dom' and local component files. The code is numbered from 1 to 8.

```
App.jsx
rc > App.jsx > App
1 import { BrowserRouter, Routes, Route } from 'react-router-dom';
2 import Classe from './components/Classe';
3 import AjouterSmartphone from './components/AjouterSmartphone';
4 import DetaillerSmartphone from './components/DetaillerSmartphone';
5 import Header from './components/Header';
6 import Footer from './components/Footer';
7 import EditerSmartphone from './components/EditSmartphone';
8
```

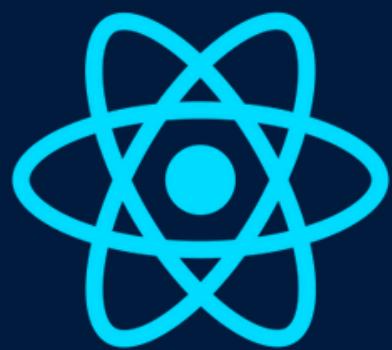


React JS

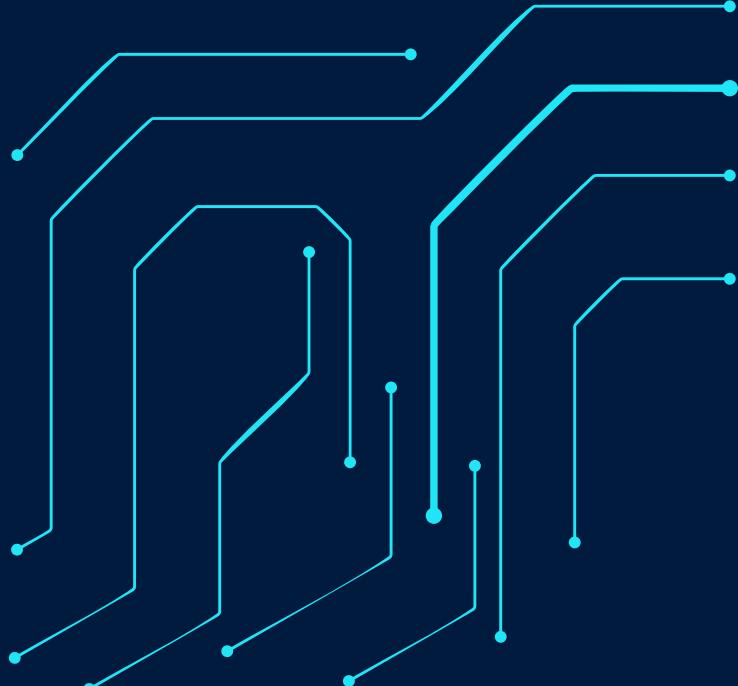


DEMONSTRATION





React JS



CONCLUSION

