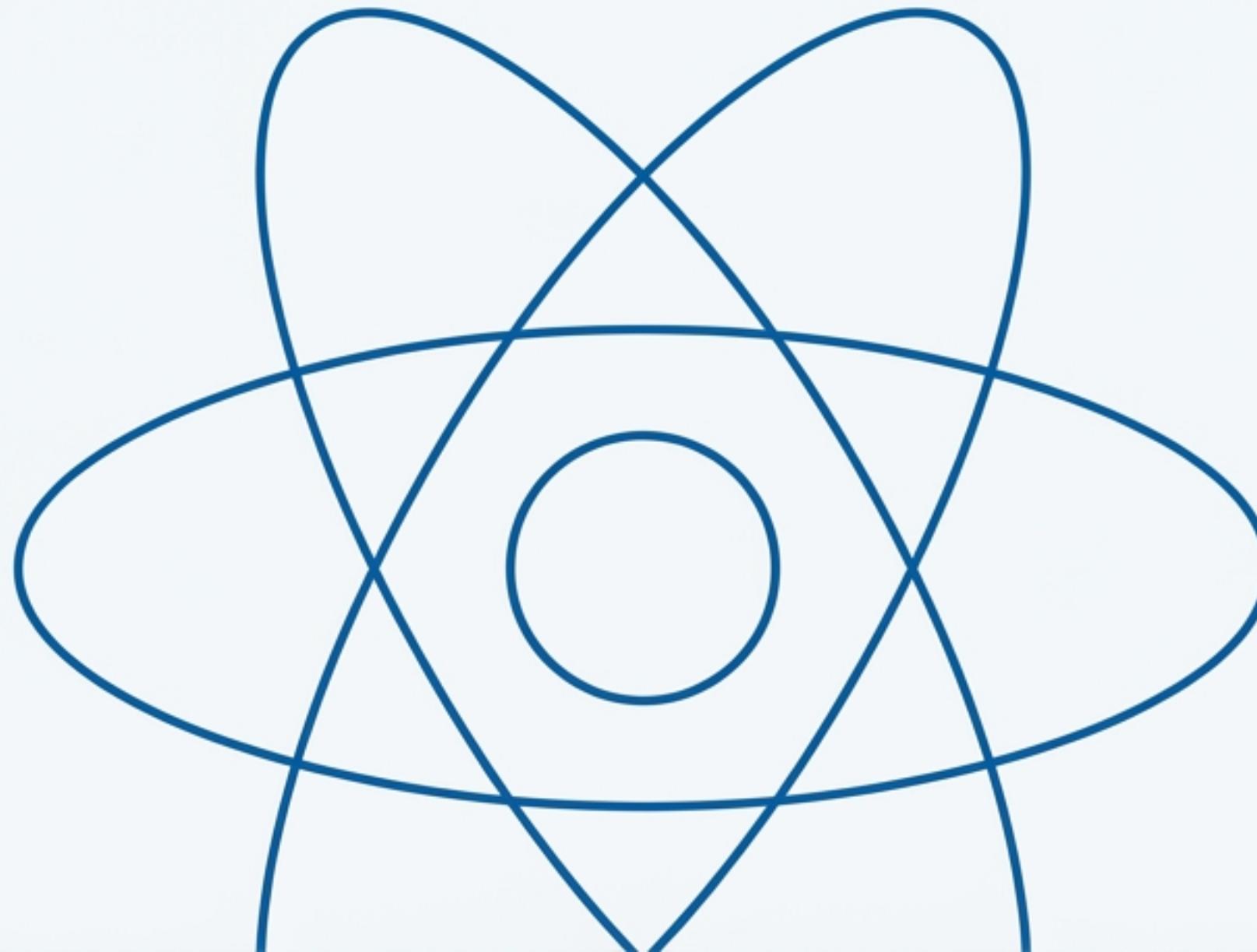


Les Fondamentaux de React

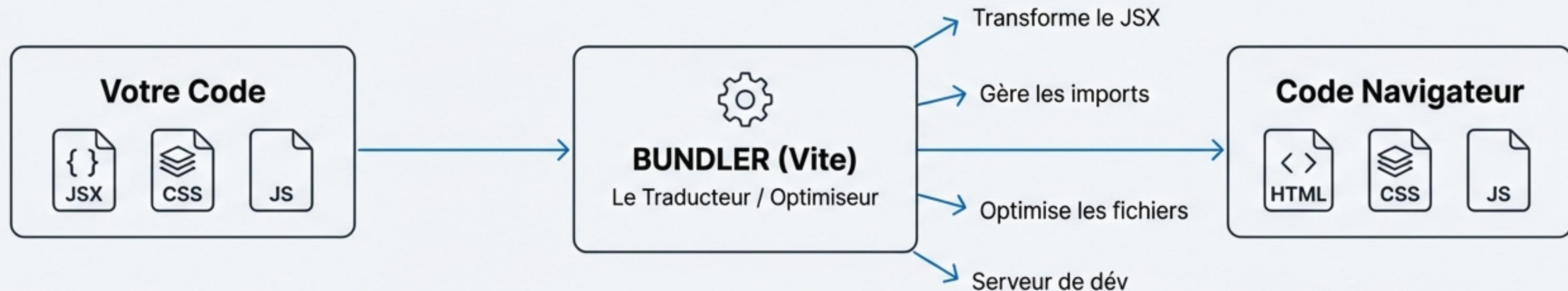
De l'idée à l'interface interactive : Votre parcours pour maîtriser les bases.



L'Écosystème Moderne : Pourquoi React a besoin d'outils

Le Principe

Votre code React utilise des syntaxes modernes (comme le JSX) que les navigateurs ne comprennent pas nativement. Un bundler est l'outil indispensable qui traduit et optimise votre projet en fichiers que n'importe quel navigateur peut exécuter.



Les Outils du Métier

Vite: Moderne, extrêmement rapide, idéal pour commencer avec React.

Webpack: L'outil historique, ultra-configurable et puissant.

Parcel: Apprécié pour son approche 'zéro configuration'.

À Retenir

Le bundler est le pont entre votre code de développeur et le code exécuté par le navigateur.

Votre Premier Projet : Anatomie d'une application React

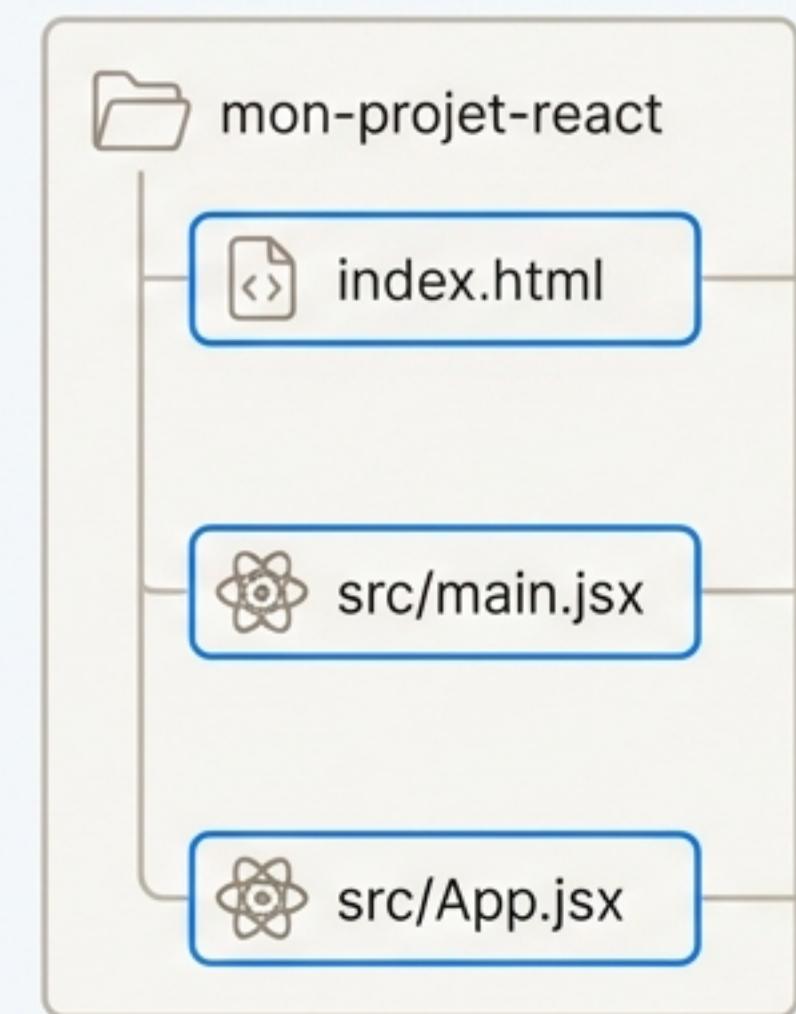
La Commande Magique (avec Vite)

```
# 1. Création du projet  
npm create vite@latest mon-projet-react -- --  
template react
```

```
# 2. Installation des dépendances  
cd mon-projet-react  
npm install
```

```
# 3. Lancement du serveur  
npm run dev
```

Le Plan de l'Application



La coquille de votre application.
Contient la balise cruciale
`<div id="root"></div>`, qui
est le point d'ancrage où React
injectera toute l'interface.

Le point d'entrée. C'est ici que
l'on dit à React de "rendre" le
composant principal `<App />` à
l'intérieur de la div `#root`.

Le cœur de votre application, le
premier composant racine à partir
duquel tout le reste est construit.

Le JSX : Quand JavaScript rencontre le HTML

Le Principe

Le JSX est une extension syntaxique pour JavaScript qui vous permet de décrire à quoi votre interface utilisateur doit ressembler. Ce n'est pas du HTML, mais une manière plus expressive et puissante d'écrire des éléments React.

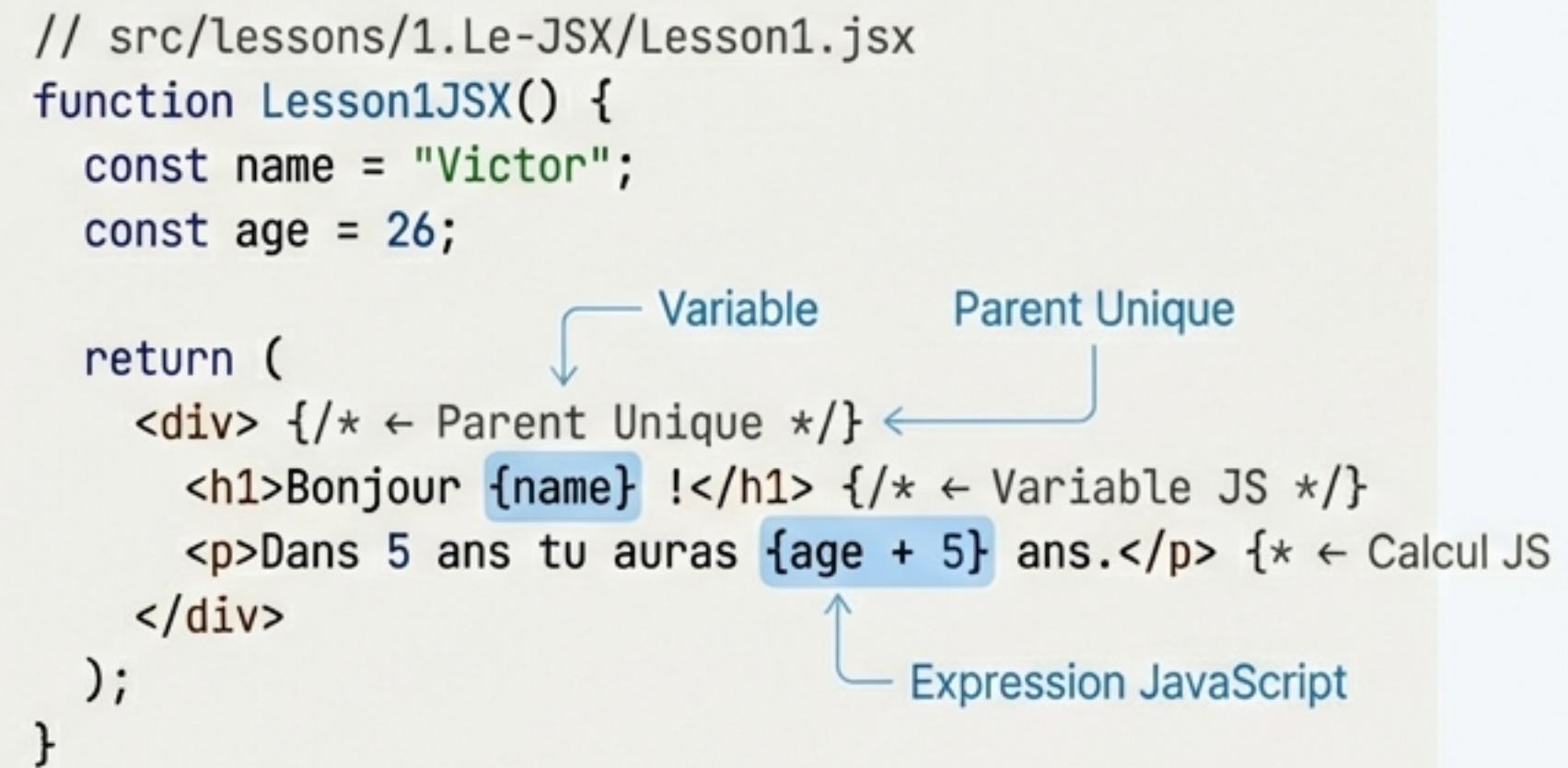
Les Règles d'Or du JSX

- **Parent Unique**:** Un composant doit toujours retourner un seul élément parent (une `div`, un `<>`, etc.).
- **La Puissance des Accolades `{}`**:** Pour injecter n'importe quelle expression JavaScript (variable, calcul, appel de fonction) directement dans votre 'HTML'.
- **Attributs en 'camelCase'**:** On utilise `className` au lieu de `class`, `onClick` au lieu de `onclick`, etc.

En Pratique

```
// src/lessons/1.Le-JSX/Lesson1.jsx
function Lesson1JSX() {
  const name = "Victor";
  const age = 26;

  return (
    <div> /* ← Parent Unique */ 
      <h1>Bonjour {name} !</h1> /* ← Variable JS */
      <p>Dans 5 ans tu auras {age + 5} ans.</p> /* ← Calcul JS
    </div>
  );
}
```



Les Composants : Penser en blocs réutilisables

Le Principe

Un composant React est une fonction JavaScript qui respecte deux règles : son nom commence par une majuscule et elle retourne du JSX. Pensez-y comme des balises HTML personnalisées et réutilisables que vous pouvez assembler pour construire des interfaces complexes.

La Recette d'un Composant

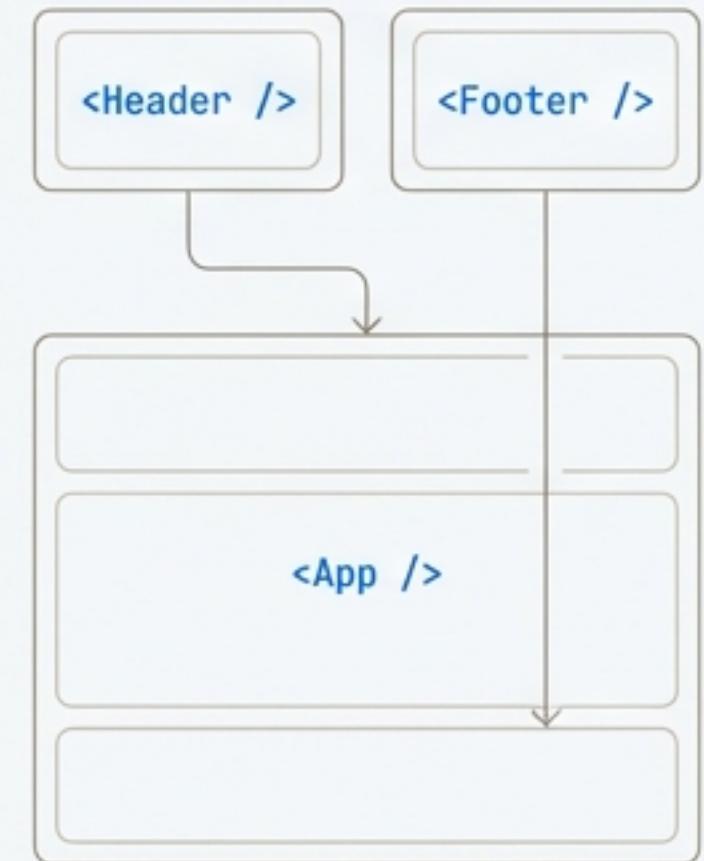
1. Créer une fonction `function MonComposant()`.
2. Retourner du JSX décrivant l'UI.
3. Exporter la fonction `export default MonComposant;`.
4. Importer et l'utiliser ailleurs comme une balise : `<MonComposant />`.

En Pratique (Assembler une page)

```
// Deux composants indépendants et réutilisables
function Header() {
  return <h1>Mon premier composant React</h1>;
}

function Footer() {
  return <footer>© 2025 Victor Besson</footer>;
}

// Le composant App les assemble
function App() {
  return (
    <>
      <Header />
      <p>Voici le contenu principal de ma page.</p>
      <Footer />
    </>
  );
}
```



Les Props : Passer des données de parent à enfant

Le Principe

Les `props` (propriétés) sont le mécanisme qui permet à un composant parent de configurer et de passer des données à ses composants enfants. Elles sont passées comme des **attributs** HTML et sont en **lecture seule** pour l'enfant.

Le Flux de Données

1. **Le Parent Donne:** Le composant parent passe les données via des attributs sur la balise du composant enfant.

```
<UserCard name="Alice" age={30} />
```

2. **L'Enfant Reçoit:** Le composant enfant reçoit ces données comme premier argument de sa fonction. On utilise souvent la **déstructuration** pour un accès direct.

```
function UserCard({ name, age }) { ... }
```

En Pratique

```
function App() {
  return (
    <div>
      <UserCard name="Victor" age={26} job="Développeur web" />
      <UserCard name="Alice" age={30} job="Designer UX" />
    </div>
  );
}
```

Parent : App.jsx

```
function UserCard({ name, age, job }) {
  return (
    <article className="user-card">
      <h2>{name}</h2>
      <p>{age} ans</p>
      <p>{job}</p>
    </article>
  );
}
```

Enfant : UserCard.jsx

Le State : La mémoire interne d'un composant

Le Principe

Si les `props` sont des données venues de l'extérieur, le `state` est la mémoire interne et privée d'un composant. C'est le `state` qui rend une application interactive. Quand le `state` change, React re-render automatiquement le composant pour refléter ce changement.

Le Hook `useState`

- **Déclaration :** const [valeur, setValeur] = useState(valeurInitiale);
- useState retourne un tableau avec deux éléments : la valeur actuelle du state, et la fonction pour la mettre à jour.
- **Mise à jour :** Appeler setValeur(nouvelleValeur) demande à React un nouveau rendu. Pour des mises à jour basées sur la valeur précédente, on utilise la forme fonctionnelle :
setCount(prevCount => prevCount + 1).

En Pratique (Un compteur)

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

function Counter() {
  const [count, setCount] = useState(0); // ← Déclaration du state

  function handleIncrement() {
    setCount(prev => prev + 1); // ← Mise à jour du state
  }

  return (
    <div>
      <p>Compteur : {count}</p>
      <button onClick={handleIncrement}>+1</button>
    </div>
  );
}
```

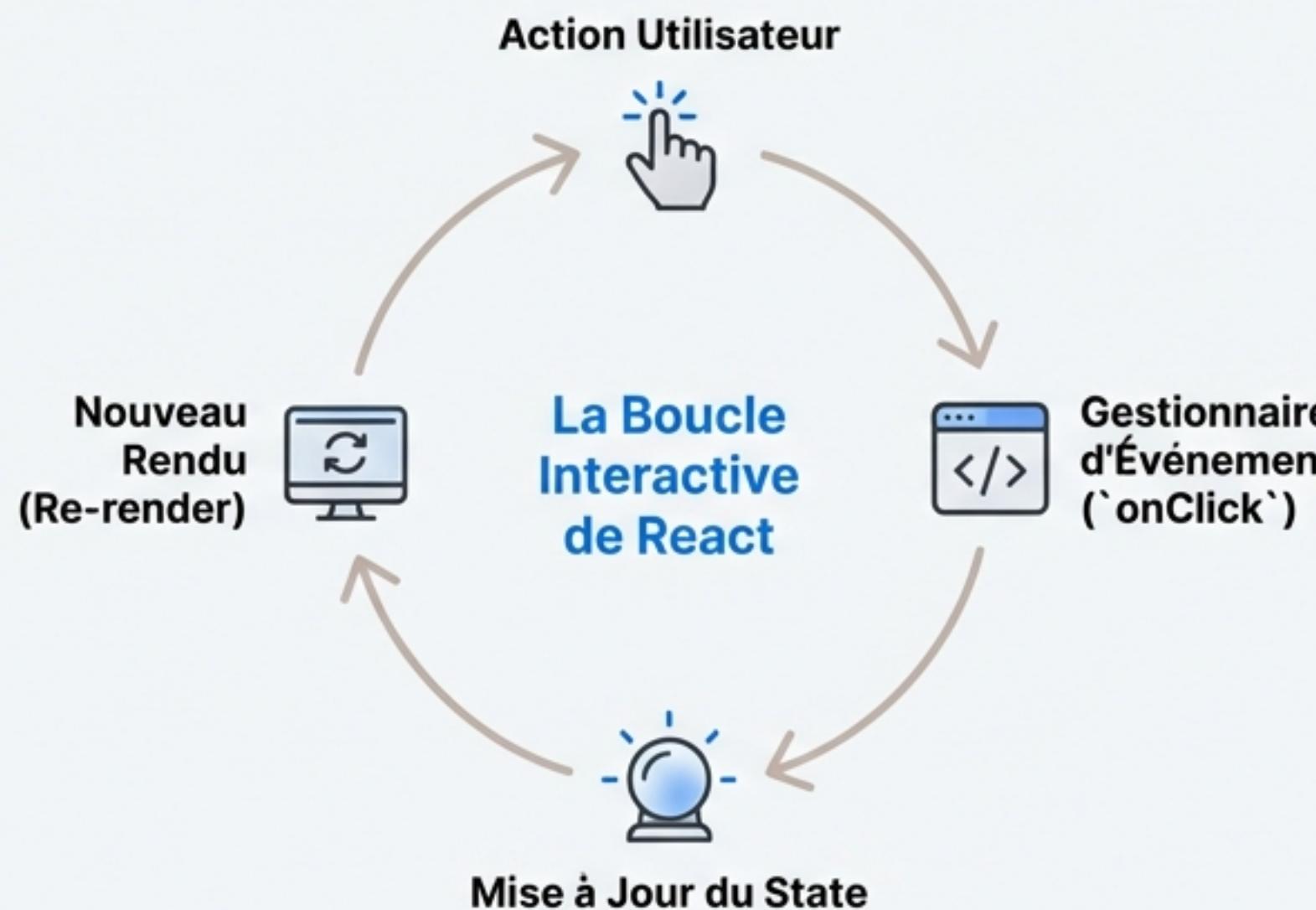
À Retenir

Props = Instructions reçues. State = Mémoire interne.

Les Événements : Écouter l'utilisateur

Le Principe

Pour rendre une application interactive, il faut réagir aux actions de l'utilisateur. En React, on attache des gestionnaires d'événements (comme `onClick` ou `onChange`) directement dans le JSX. Ces gestionnaires sont des fonctions qui seront appelées au bon moment.



En Pratique (Un champ de saisie)

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

function InteractiveInput() {
  const [inputValue, setInputValue] = useState("");

  // Ce gestionnaire met à jour le state à chaque frappe
  function handleInputChange(event) {
    setInputValue(event.target.value);
  }

  function handleButtonClick() {
    alert("Valeur soumise : " + inputValue);
  }

  return (
    <div>
      <input type="text" onChange={handleInputChange} />
      <button onClick={handleButtonClick}>Afficher la valeur</button>
    </div>
  );
}
```

Afficher des données : Listes et Images

Le Principe

Les applications affichent rarement des données statiques. On utilise des méthodes JavaScript standard comme ` `.map()` pour transformer des tableaux de données en listes d'éléments JSX.

Créer une Liste avec ` .map()`

```
const users = [
  { id: 1, name: "Victor", role: "Admin" },
  { id: 2, name: "Alice", role: "User" },
];

function UserList() {
  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name} - {user.role}</li>
      ))}
    </ul>
  );
}
```

Indispensable pour la performance !

- On part d'un tableau de données.
- On utilise `{tableau.map(item => <MonComposant ... />)}` pour générer un composant pour chaque item.
- **Crucial:** On ajoute une prop `key` unique et stable (souvent `item.id`) à chaque élément de la liste. `key` aide React à optimiser les mises à jour.

Utiliser des Images

1. **'Image Locale'**: Importer l'image depuis `/src` comme un module. React (via Vite) s'occupe du chemin.

```
import localAvatar from "./assets/avatar.png";
<img src={localAvatar} />
```

2. **'Image Externe'**: Utiliser directement l'URL de l'image.

```

```

N'oubliez jamais l'attribut `alt` pour l'accessibilité !

Le Rendu Conditionnel : L'art d'afficher au bon moment

Le Principe

Vous aurez souvent besoin d'afficher ou de masquer des éléments d'interface en fonction de certaines conditions (état de connexion, données disponibles, etc.). React offre plusieurs patrons pour cela.

Votre Boîte à Outils pour les Conditions

La Méthode `if`

La plus lisible et explicite. Idéale pour des logiques complexes ou pour retourner des composants complètement différents. On prépare le JSX dans une variable avant le `return`.

```
function StatusMessage({ isLoggedIn }) {  
  if (isLoggedIn) {  
    return <p>Bienvenue !</p>;  
  }  
  return <p>Veuillez vous connecter.</p>;  
}
```

L'Opérateur Tertiaire `?:`

Parfait pour des choix simples `if/else` directement dans le JSX. La syntaxe est `condition ? <JSXsiVrai /> : <JSXsiFaux />`.

```
function ProductStatus({ inStock }) {  
  return (  
    <p>Statut : {inStock ? 'En stock' :  
      'Rupture'}</p>  
  );  
}
```

Le Court-Circuit `&&`

Le plus concis pour afficher un élément *seulement si* une condition est vraie. La syntaxe est `condition && <JSX />`. Si la condition est fausse, rien n'est affiché.

```
function Notification({ count }) {  
  return (  
    <div>  
      {count > 0 && <p>Vous avez {count}  
        notifications.</p>}  
    </div>  
  );  
}
```

Style et Dynamisme : CSS et classes conditionnelles

Le Principe

L'application des styles en React se fait principalement via l'attribut `className`. En combinant `className` avec le `state`, `state`, on peut créer des interfaces visuellement dynamiques qui réagissent aux actions de l'utilisateur.

Appliquer une Classe CSS

- Le CSS est souvent défini dans un fichier `*.css` et importé.
- On utilise `className="ma-classe autre-classe\`` pour appliquer les styles.

Rendre les Classes Dynamiques

- On utilise un `state` (souvent un booléen comme `isActive` ou `isDark`).
- On construit la chaîne de `className` en utilisant un opérateur ternaire ou des template literals.

En Pratique (Un toggle Dark Mode)

```
/* lesson12.css */  
.box { border: 1px solid #ddd; padding: 1rem; }  
.box--dark { background-color: #111827; color: #f9fafb; }  
(CSS)
```

```
import { useState } from "react";  
import "./lesson12.css";  
  
(JSX)
```

```
function DarkModeToggle() {  
  const [isDark, setIsDark] = useState(false);  
  
  const boxClassName = `box ${isDark ? "box--dark" : ""}`; → Ici, on construit la  
  chaîne de classes.  
  Si `isDark` est  
  `true`, on  
  ajoute la classe  
  `box--dark`.  
  
  return (  
    <div>  
      <button onClick={() => setIsDark(prev => !prev)}>  
        Passer en mode {isDark ? "clair" : "sombre"}  
      </button>  
      <div className={boxClassName}>  
        <p>Je change de style en fonction du state.</p>  
      </div>  
    </div>  
  );  
}
```

L'Échappatoire : `useRef` et la manipulation directe

Le Principe

Le hook `useRef` offre une 'référence' vers un élément du DOM. Il est principalement utilisé dans deux cas :

- 1. Accéder à un élément DOM:** Pour effectuer des actions que React ne gère pas nativement, comme donner le focus à un `input`.
- 2. Stocker une valeur mutable:** Pour garder une valeur (comme un ID de `setInterval`) qui peut changer sans pour autant provoquer un nouveau rendu du composant.

En Pratique (Donner le focus à un input)

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

function FocusInput() {
  const inputRef = useRef(null); // 1. Créer la ref —————— 1. Création

  function handleFocusClick() {
    // 3. Utiliser la ref pour appeler une méthode du DOM
    inputRef.current?.focus(); —————— 3. Utilisation → | I
  }

  return (
    <div>
      {/* 2. Attacher la ref à l'élément DOM */}
      <input ref={ inputRef } type="text" /> —————— 2. Attachement
      <button onClick={handleFocusClick}>Donner le focus</button>
    </div>
  );
}
```

La Différence Clé

- `useState`: Une modification de la valeur **provoque un re-render**.
- `useRef`: Une modification de la valeur (`ref.current = ...`) **ne provoque pas de re-render**.

La Maîtrise des Formulaires : Contrôlé vs. Non Contrôlé

Le Principe

Il existe deux approches pour gérer les données des champs de formulaire (`input`, `textarea`, etc.). Le choix dépend du niveau de contrôle dont vous avez besoin.

Input Contrôlé (La voie React)

Philosophie: React est la **source unique de vérité**. La valeur de l'input est stockée dans le `state`.

Mécanisme

1. La prop `value` de l'input est liée à une variable de `state`.
2. La prop `onChange` est utilisée pour mettre à jour ce `state` à chaque frappe.

Avantages

Validation instantanée, formatage en temps réel, bouton de soumission désactivable facilement.

```
const [email, setEmail] = useState("");  
  
<input  
  type="email"  
  value={email} // ← La valeur vient du state  
  onChange={(e) => setEmail(e.target.value)} // ← Chaque frappe met à jour le >
```

Input Non Contrôlé (La voie DOM)

Philosophie: Le DOM gère lui-même la valeur de l'input, comme en HTML classique.

Mécanisme

1. On attache une `ref` (`useRef`) à l'élément `input`.
2. On lit la valeur via `ref.current.value` uniquement lorsque c'est nécessaire (ex: à la soumission du formulaire).

Avantages

Moins de code, plus simple pour des formulaires basiques, intégration facile avec des bibliothèques non-React.

```
const emailRef = useRef(null);  
  
function handleSubmit(e) {  
  const email = emailRef.current.value; // ← On lit la valeur à la demande  
  // ...  
}  
  
<input type="email" ref={emailRef} />
```

Votre Parcours de Développeur React

De l'Idée à l'Interface : Vous avez acquis les compétences fondamentales pour construire des applications web modernes.
Récapitulons le chemin parcouru.



Prochaines Étapes : Au-delà des fondamentaux

Le Principe

Vous avez maintenant des bases solides. Le voyage pour devenir un expert React continue en explorant les outils et concepts qui résolvent des problèmes plus complexes.

Les Chemins à Explorer



React Router

Pour créer des applications multi-pages avec une navigation côté client.



`useEffect`

Le hook pour gérer les 'effets de bord' : interagir avec des API externes, s'abonner à des événements, etc.



Context API / Redux

Pour gérer un 'état global' partagé à travers toute votre application sans passer des props à travers de nombreux niveaux.



React Query / TanStack Query

Pour simplifier radicalement la récupération, la mise en cache et la synchronisation des données provenant d'un serveur.

La meilleure façon d'apprendre est de pratiquer.

Continuez à construire !