

# RÉFÉRENCE RAPIDE REACT

Composants, hooks, état, effets, patterns

## Bases JSX

### Expressions & Attributs

```
const name = "Alice";
const el = <h1>Hello, {name}</h1>;
const img = <img src={url} alt="photo" />;
```

### Règles JSX

<b>{expression}</b>	Intégrer toute expression JS
<b>className</b>	À utiliser à la place de `class`
<b>htmlFor</b>	À utiliser à la place de `for`
<b>style={{color: 'red'}}</b>	Styles inline sous forme d'objet
<b>&lt;Component /&gt;</b>	Balises auto-fermantes obligatoires
<b>&lt;&gt; ... &lt;/&gt;</b>	Fragment (sans nœud DOM supplémentaire)

## Composants

### Composants fonctionnels

```
function Greeting({ name }) {
  return <h1>Hello, {name}</h1>;
}

// Arrow function variant
const Greeting = ({ name }) => (
  <h1>Hello, {name}</h1>
);
```

### Props

```
function Card({ title, children }) {
  return (
    <div className="card">
      <h2>{title}</h2>
      {children}
    </div>
  );
}

<Card title="Welcome">
  <p>Content here</p>
</Card>
```

### Props par défaut

```
function Button({ label = "Click me", onClick }) {
  return <button onClick={onClick}>{label}</button>;
}
```

## État (useState)

### État de base

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

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

### Mises à jour fonctionnelles

```
setCount(prev => prev + 1); // use prev state
setItems(prev => [...prev, newItem]); // arrays
setUser(prev => ({...prev, name: "Bob"})); // objects
```

### Règles d'état

<b>Mises à jour immuables</b>	Ne jamais muter l'état directement
<b>Batch asynchrone</b>	Les mises à jour peuvent être groupées
<b>Forme fonctionnelle</b>	Utiliser `prev` quand l'état précédent est nécessaire

## Effets (useEffect)

### Patterns d'effet

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

// Run on every render
useEffect(() => { /* ... */ });

// Run once on mount
useEffect(() => { /* ... */ }, []);

// Run when deps change
useEffect(() => { /* ... */ }, [count]);

// Cleanup on unmount
useEffect(() => {
  const id = setInterval(tick, 1000);
  return () => clearInterval(id);
}, []);
```

## Listes & Clés

```
function TodoList({ items }) {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.text}</li>
      ))}
    </ul>
  );
}
```

Les clés doivent être des IDs stables et uniques — éviter l'index comme clé

## Gestion des événements

### Événements

```
<button onClick={handleClick}>Click</button>
<button onClick={() => handleDelete(id)}>Del</button>
<input onChange={e => setVal(e.target.value)} />
<form onSubmit={e => {
  e.preventDefault();
  handleSubmit();
}}>
```

### Événements courants

<b>onClick</b>	Clic de souris
<b>onChange</b>	Changement de valeur d'un champ
<b>onSubmit</b>	Soumission de formulaire
<b>onKeyDown</b>	Appui sur une touche

<b>onFocus / onBlur</b>	Focus obtenu / perdu
<b>onMouseEnter</b>	Souris entre dans l'élément

## Rendu conditionnel

```
// Ternary
{isLoggedIn ? <Dashboard /> : <Login />}

// Logical AND (short-circuit)
{hasError && <ErrorBanner />}

// Early return
function Page({ user }) {
  if (!user) return <Login />;
  return <Dashboard user={user} />;
}
```

## Hooks

### useRef

```
const inputRef = useRef(null);
// Access: inputRef.current.focus();
<input ref={inputRef} />
```

useRef conserve les valeurs entre les rendus sans déclencher de re-rendu

### useMemo & useCallback

```
// Memoize expensive computation
const sorted = useMemo(
  () => items.sort(compareFn),
  [items]
);

// Memoize callback
const handleClick = useCallback(
  () => setCount(c => c + 1),
  []
);
```

### useContext

```
import { useContext } from "react";
const value = useContext(MyContext);
```

## Hooks personnalisés

```
function useLocalStorage(key, initial) {
  const [value, setValue] = useState(() => {
    const saved = localStorage.getItem(key);
    return saved ? JSON.parse(saved) : initial;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}
```

```
// Usage
const [name, setName] = useLocalStorage("name", "");
```

Les hooks personnalisés doivent commencer par `use`

## API Context

### Créer & Fournir

```
import { createContext, useContext } from "react";
const ThemeCtx = createContext("light");

function App() {
  return (
    <ThemeCtx.Provider value="dark">
      <Page />
    </ThemeCtx.Provider>
  );
}
```

### Consommer

```
function Page() {
  const theme = useContext(ThemeCtx); // "dark"
  return <div className={theme}>...</div>;
}
```