React + TypeScript - Cheat Sheet

useState - État local

```tsx
const [state, setState] = useState<Type>(init);
const [count, setCount] = useState(0);

// Update
setCount(5);
setCount(prev => prev + 1);

// Object (ALWAYS spread)
setUser({...user, name: "Alice"});

// Array (ALWAYS immutable)
setItems([...items, "new"]);
setItems(items.filter(i => i !== "old"));
setItems(items.map(i => i === "a" ? "b" : i));
```

useEffect - Effets de bord

```tsx
// Mount only
useEffect(() => {
  console.log("Mounted");
}, []);

// When dep changes
useEffect(() => {
  fetchData(userId);
}, [userId]);

// Cleanup
useEffect(() => {
  const timer = setInterval(tick, 1000);
  return () => clearInterval(timer);
}, []);

// Fetch pattern
useEffect(() => {
  let ignore = false;
  fetch(`/api/user/${id}`)
    .then(r => r.json())
    .then(data => !ignore && setUser(data));
  return () => { ignore = true; };
}, [id]);
```

useContext - Contexte global

```tsx
// 1. Create context
const ThemeContext = createContext<Type | null>(null);

// 2. Provider
export const ThemeProvider = ({children}) => {
  const [theme, setTheme] = useState("light");
  const toggle = () => setTheme(p =>
    p === "light" ? "dark" : "light");

  return <ThemeContext.Provider value={{theme, toggle}}>
    {children}
  </ThemeContext.Provider>;
};

// 3. Hook
export const useTheme = () => {
  const ctx = useContext(ThemeContext);
  if (!ctx) throw new Error("Missing provider");
  return ctx;
};

// 4. Usage
const {theme, toggle} = useTheme();
```

JSX/TSX Syntax

```tsx
<div>{user.name}</div>
<div>{isLoading ? "Loading..." : "Done"}</div>

// Attributes (camelCase)
<div className="card" onClick={handleClick}>
<input value={name} onChange={e => setName(e.target.value)} />
<label htmlFor="email">Email</label>

// Conditional
{isLoading && <Spinner />}
{error ? <Error /> : <Content />}

// Lists (ALWAYS key)
{items.map(item => (
  <div key={item.id}>{item.name}</div>
))}

// Fragment
<><Header /><Main /></>

// Style inline
<div style={{color: "red", fontSize: 16}}>
```

Props vs State

```tsx
// PROPS (readonly, comes from parent)
interface ButtonProps {
  label: string;
  onClick: () => void;
  disabled?: boolean;
}

const Button = ({label, onClick, disabled = false}: ButtonProps) => (
  <button onClick={onClick} disabled={disabled}>
    {label}
  </button>
);

// Usage
<Button label="Click" onClick={handleClick} />

// STATE (editable, internally to the component)
const [count, setCount] = useState(0);

// Props ↓ / Events ↑
function Parent() {
  const [data, setData] = useState("");
  return <Child value={data} onChange={setData} />;
}
```

Props	State
Data from parent	Internal component data
Read-only	Mutable
Configuration values	Data that changes over time

React Router

```tsx
import { BrowserRouter, Routes, Route, Link,
  useNavigate, useParams } from "react-router-dom";

<BrowserRouter>
  <Link to="/">Home</Link>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/users/:id" element={<User />} />
    <Route path="*" element={<NotFound />} />
  </Routes>
</BrowserRouter>

// Hooks
const navigate = useNavigate();
navigate("/about");

const {id} = useParams<{id: string}>();

const [params, setParams] = useSearchParams();
const q = params.get("q");
```

🌐 i18n (react-i18next)

```ts
// i18n.ts
import i18n from "i18next";
import { initReactI18next } from "react-i18next";

i18n.use(initReactI18next).init({
  resources: {
    en: { translation: { welcome: "Welcome {{name}}" }},
    fr: { translation: { welcome: "Bienvenue {{name}}" }}
  },
  lng: "fr", fallbackLng: "en"
});

// Component
const {t, i18n} = useTranslation();
<h1>{t("welcome", {name: "Alice"})}</h1>
<button onClick={() => i18n.changeLanguage("en")}>EN</button>
```

🎯 TypeScript Types

```ts
interface Props {
  title: string;
  count?: number;
  onSave: (data: string) => void;
  children: React.ReactNode;
}

// Events
(e: React.ChangeEvent<HTMLInputElement>) => {}
(e: React.MouseEvent<HTMLButtonElement>) => {}
(e: React.FormEvent<HTMLFormElement>) => {}

// Refs
const ref = useRef<HTMLInputElement>(null);
ref.current?.focus();
```

⚡ Best Practices

- ✅ PascalCase: `const UserCard = () => {}`
- ✅ One file = one component
- ✅ Destructure props
- ✅ Explicit types: `useState<User[]>([])`
- ✅ Immutability: `setItems([...items, x])`
- ✅ Exhaustive deps in useEffect
- ✅ Cleanup effects when needed
- ✅ Unique keys in lists
- ❌ NEVER mutate: `items.push(x)`
- ❌ NEVER: `user.name = "Bob"`

🚀 npm Commands

```bash
npm create vite@latest app -- --template react-ts
npm install react-router-dom
npm install react-i18next i18next

npm run dev      # Dev :5173
npm run build    # Prod → dist/
npm run preview  # Test build
```

📋 Common Patterns

```ts
// Custom hook
function useLocalStorage(key: string, init: string) {
  const [val, setVal] = useState(() =>
    localStorage.getItem(key) ?? init);

  useEffect(() => {
    localStorage.setItem(key, val);
  }, [key, val]);

  return [val, setVal] as const;
}

// Controlled input
<input value={val} onChange={e => setVal(e.target.value)} />

// Form
const [form, setForm] = useState({name: "", email: ""});
<input value={form.name}
  onChange={e => setForm({...form, name: e.target.value})} />
```

🔑 Hooks Summary

Hook	Purpose
useState	Local state
useEffect	Side effects
useContext	Global state
useRef	DOM refs
useMemo	Memoize values
useCallback	Memoize functions

⚠️ Common Pitfalls

```ts
// ❌ Missing deps
useEffect(() => { fetch(url); }, []);

// ❌ Mutating state
items.push(x); setItems(items);

// ❌ Missing key
{items.map(x => <div>{x}</div>)}

// ❌ Stale closure
useEffect(() => {
  setInterval(() => setCount(count + 1), 1000);
}, []);

// ✅ Fix with functional update
useEffect(() => {
  const id = setInterval(() => setCount(c => c + 1), 1000);
  return () => clearInterval(id);
}, []);
```

🎨 Styling

```ts
// Inline
<div style={{color: "red", marginTop: 10}}>

// Class
<div className="card active">

// Conditional
<div className={isActive ? "active" : "inactive"}>
<div className={`card ${isActive ? "active" : ""}`}>
```

🆘 Debug

```ts
console.log("State:", state);
debugger;

// Re-render tracking
useEffect(() => {
  console.log("Re-rendered");
});

// React DevTools (Chrome extension)
// Components tab: props/state
// Profiler tab: performance
```

📚 Resources

- **React:** react.dev
- **TypeScript:** typescriptlang.org
- **React Router:** reactrouter.com
- **i18next:** react.i18next.com
- **Vite:** vitejs.dev