React: Componentes, Props y Hooks

Diego Muñoz

09 de noviembre de 2025

Introducción

- Formalizar conceptos centrales de React.
- Construir componentes reutilizables.
- Practicar `props` y el hook `useState`.
- Bases sólidas antes de entrar a temas avanzados (ej: Context, Router).

¿Qué es un componente?

- Una **función de JavaScript** que devuelve “HTML” (JSX).
- Debe iniciar con **mayúscula**.
- Puede recibir datos como **props**.
- Se pueden componer: un componente dentro de otro.

```
function Header() {
  return <h1>Mi App</h1>;
}

function Footer() {
  return <small>© 2025</small>;
}
```

Composición de componentes

```
function App() {
  return (
    <div>
      <Header />
      <p>Contenido principal</p>
      <Footer />
    </div>
  );
}
```

Props

- Los **props** son como los parámetros de una función.
- Son **inmutables**: no deben modificarse dentro del componente.

```
function Card({ title, body }) {
  return (
    <div className="card">
      <h3>{title}</h3>
      <p>{body}</p>
    </div>
  );
}
```

Props

- Los **props** son como los parámetros de una función.
- Son **inmutables**: no deben modificarse dentro del componente.

```
function App() {
  return (
    <div>
      <Card title="Intro a JS" body="Variables y funciones" />
      <Card title="React" body="Estado y componentes" />
    </div>
  );
}
```

Estado con useState

- `useState` crea un valor **reactivo** y una función para actualizarlo.
- React vuelve a renderizar el componente cuando el estado cambia.

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

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

Múltiples instancias

- Cada componente mantiene **su propio estado independiente**.
- Props inicializan, pero luego el estado se maneja dentro.
- **NUNCA** usar `useState` fuera de un componente.

```
function App() {
  return (
    <div>
      <Counter initial={0} />
      <Counter initial={10} />
    </div>
  );
}
```

Listas con map

- Cada elemento necesita una prop **key** única.

```
const tasks = [
  { id: 1, text: "Leer", done: false },
  { id: 2, text: "Ejercitar", done: true }
]; # Base de Datos Falsa
```

Listas con map

- Podemos usar `array.map` para renderizar muchos elementos.

```
function App() {
  return (
    <div>
      {tasks.map(t => (
        <div key={t.id}>
          {t.done ? "Listo" : "Pendiente"} {t.text}
        </div>
      ))}
    </div>
  );
}
```

Eventos

- Los eventos en React se escriben como props (`onClick`, `onChange`).
- Usan *camelCase*.
- Se pasan funciones.

```
function Button({ text, onClick }) {
  return <button onClick={onClick}>{text}</button>;
}

function App() {
  const sayHi = () => alert("Hola!");
  return <Button text="Saludar" onClick={sayHi} />;
}
```

Prop drilling

- Cuando un dato debe viajar de un componente **padre** hasta un **nieto** pasando por hijos intermedios.
- A veces obliga a pasar props que no se usan en el medio.
- Problema común en apps reales → más adelante se resuelve con **Context**.

Prop drilling

```
function Child({ name }) {
  return <p>Hola {name}</p>;
}

function Parent({ name }) {
  return <Child name={name} />;
}

function App() {
  return <Parent name="Ana" />;
}
```

Lifting state up

- Cuando varios hijos necesitan compartir un dato.
- El **estado se define en el padre** y se pasa hacia abajo como props.
- Los hijos avisan cambios llamando funciones que vienen del padre.

Lifting state up

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

Buenas prácticas iniciales

1. **Componentes pequeños** y reutilizables.
2. **Props inmutables**, no modificarlos.
3. **useState** solo dentro de componentes (no en funciones comunes).
4. **Keys únicas** al renderizar listas.
5. **JSX legible**: indentar y separar bloques.
6. **Documentar** componentes complejos con comentarios.

Proyecto de ejemplo

Referir a ejemplo en GitHub.

Resumen

- **Componentes**: funciones que devuelven JSX.
- **Props**: parámetros inmutables para personalizar componentes.
- **useState**: manejar estado local.
- **Eventos**: funciones que reaccionan a la interacción.
- **Prop drilling**: props que viajan en cadena → motivación para aprender Context más adelante.
- **Lifting state up:** estado sube al padre para ser compartido por varios hijos.